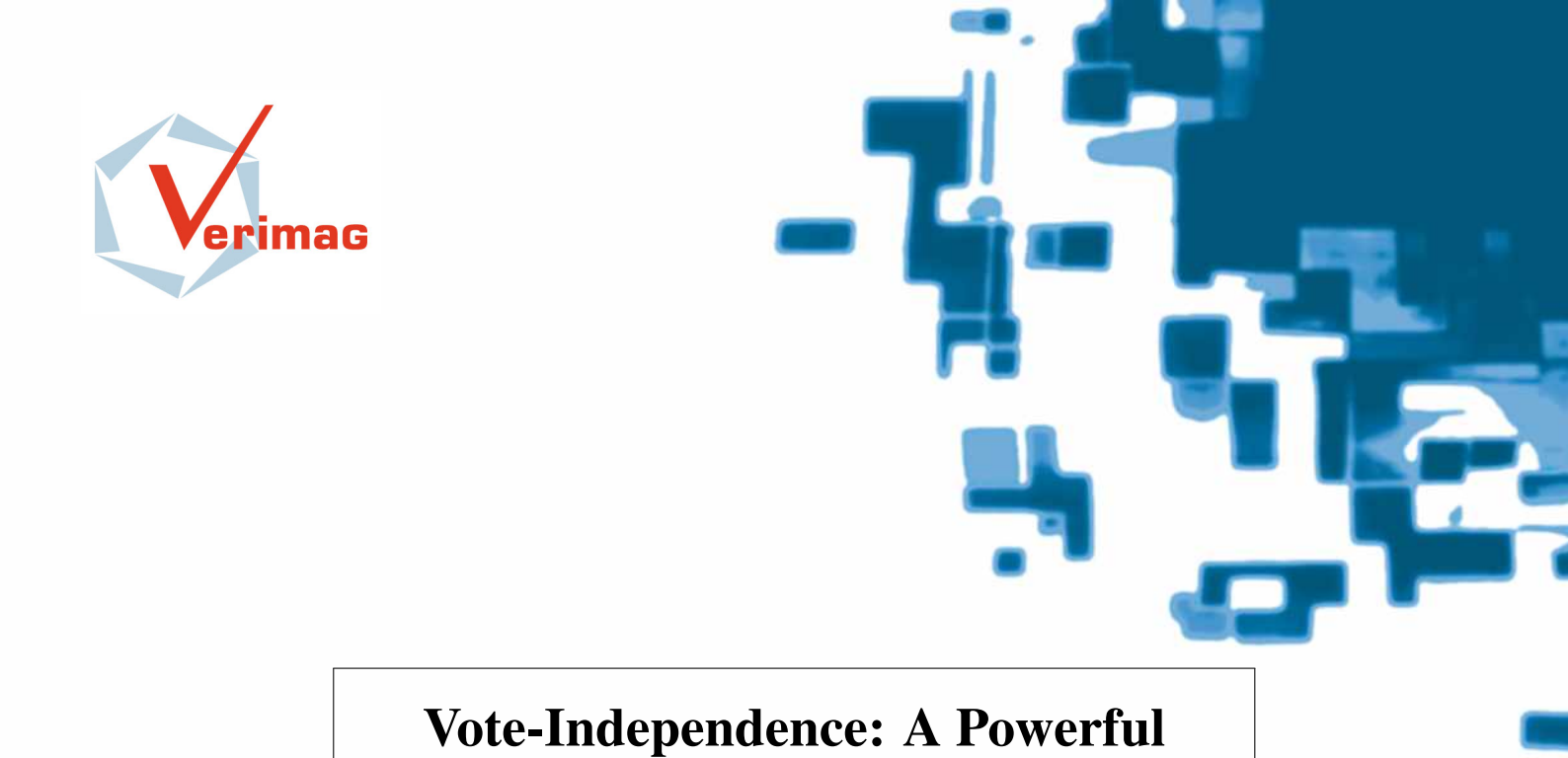
Vote-Independence: A Powerful Privacy Notion for Voting Protocols

Jannik Dreier, Pascal Lafourcade, Yassine Lakhnech



Vote-Independence: A Powerful Privacy Notion for Voting Protocols

Jannik Dreier, Pascal Lafourcade, Yassine Lakhnech

Verimag Research Report n^o TR-2011-8

April 19, 2011
Updated July 13, 2011

Reports are downloadable at the following address
http://www-verimag.imag.fr

Unité Mixte de Recherche 5104 CNRS - INPG - UJF
Centre Equation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

Vote-Independence: A Powerful Privacy Notion for Voting Protocols

Jannik Dreier, Pascal Lafourcade, Yassine Lakhnech

April 19, 2011
Updated July 13, 2011

Abstract

Recently an attack on ballot privacy in Helios has been discovered [21], which is essentially based on copying other voter's votes. To capture this and similar attacks, we extend the classical threat model and introduce a new security notion for voting protocols: Vote-Independence. We give a formal definition and analyze its relationship to established privacy properties such as Vote-Privacy, Receipt-Freeness and Coercion-Resistance. In particular we show that even Coercion-Resistant protocols do not necessarily ensure Vote-Independence.

Keywords: Electronic Voting, Privacy, Anonymity, Security, Formal Verification, Coercion-Resistance, Receipt-Freeness

How to cite this report:

```
@techreport {TR-2011-8,
  title = {Vote-Independence: A Powerful Privacy Notion for Voting Protocols},
  author = {Jannik Dreier, Pascal Lafourcade, Yassine Lakhnech},
  institution = {{Verimag} Research Report},
  number = {TR-2011-8},
  year = {2011}
}
```

1 Introduction

Electronic voting schemes are systems that allow casting and tallying votes using machines. This promises to improve efficiency by providing results faster, using less personnel or adding comfort (e.g. the possibility to vote from home). However the recent use of commercial electronic voting systems for presidential or general elections in many countries has spread controversy on security issues [5, 6, 18, 23]. Primary concerns are verifiability (the possibility to verify the elections's outcome, i.e. to check if all votes have been counted correctly) and privacy (i.e. anonymity of the voter, secrecy of the vote). To address this issues, many different protocols have been developed to fulfill security requirements such as

- *Eligibility*: Only the registered voters can vote, and nobody can vote more than once.
- *Fairness*: No preliminary results are available which could influence other voters' decisions.
- *Individual Verifiability*: Each voter can check whether his vote was counted correctly.
- *Universal Verifiability*: Anybody can verify that the announced result corresponds to the sum of all votes.
- *Vote-Privacy*: The votes are kept private.
- *Receipt-Freeness*: A voter cannot construct a receipt which allows him to prove to a third party that he voted for a certain candidate. This is to prevent vote-buying.
- *Coercion-Resistance*: Even when a voter interacts with a coercer during the entire voting process, the coercer cannot be sure whether the voter followed his instructions or actually voted for another candidate.
- *Robustness*: The protocol should be able to tolerate a certain number of misbehaving voters.

A common aim is to verify these properties using formal models and definitions. This concerns privacy properties (privacy, receipt-freeness and coercion-resistance) [7, 8, 16], election verifiability [14, 22], or both [11, 12, 13]. We concentrate on privacy-type properties of voting protocols (i.e. Vote-Privacy, Receipt-Freeness and Coercion-Resistance).

While analyzing privacy in Helios [2], a web based voting system, B. Smyth and V. Cortier [20, 21] recently discovered an attack based on the possibility for an attacker to copy another voter's vote and to submit it as his own. If the number of participating voters is small or if a noticeable fraction of voters can be corrupted, this can break privacy as the contents of the vote can be inferred from the published election outcome. For example in the case of three voters (two honest ones and one under the control of the attacker), the attacker can try to copy the vote of the first honest voter. The candidate chosen by the targeted voter will then have at least two votes, and can thus be recognized in the official outcome. This reveals the content of the targeted vote.

Our Contributions Based on this attack, we extend the established threat model and give a formal definition for the notion of "Vote-Independence (VI)" in the applied pi calculus. We show that our definition of "Vote-Independence" implies Vote-Privacy as defined in the literature [7] and that the concept can be generalized to improve Receipt-Freeness and Coercion Resistance as well. We define "Vote-Independence with passive Collaboration (VI-PC)" (which corresponds to Vote-Independence in same setting as Receipt-Freeness, i.e. with passive collaboration of the voter) and "Vote-Independence with active Collaboration (VI-AC)" (which corresponds to Coercion Resistance). We prove the hierarchy of the definitions and illustrate each level with a real world example: the protocol by Fujioka et al. [10] provides Vote-Independence (VI), the protocol due to Okamoto [19] ensures VI-PC, and the protocol by Bohli et al. [4] guarantees VI-AC. We also show that even Coercion-Resistant protocols may not ensure Vote-Independence (i.e. that our definitions are strictly stronger than the usual privacy notions) by analyzing the protocol by Lee at al. [15].

Outline of the Paper The remainder of the paper is structured as follows. In Section 2, we recall the applied pi calculus and the standard privacy definitions. In Section 3 we elaborate our definitions of Vote-Independence. Then we analyze the hierarchy of our definitions, the relation to standard privacy properties and discuss several examples in Section 4. Finally, we conclude and discuss future work.

A short version of this report has been presented at FPS 2011 [9].

2 Preliminaries

In this section we introduce the applied pi calculus, define our model of voting processes and recall existing privacy definitions.

2.1 The Applied Pi Calculus

We use the applied pi calculus [1] to model our security properties and the protocols to analyze. The calculus is an abstract language to describe concurrent processes and interactions, and is supported by the tool "ProVerif" [3].

The calculus consists of *names* (which typically correspond to data or channels), *variables*, and a *signature* Σ of *function symbols* which can be used to build *terms*. Functions typically include encryption and decryption (for example $\mathtt{enc}(message, key)$, $\mathtt{dec}(message, key)$), hashing, signing etc. Terms are correct (i.e. respecting arity and sorts) combinations of names and functions. We distinguish the type "channel" from other *base* types. To model equalities we use an equational theory E which defines a relation $=_E$. A classical example for symmetric encryption is $\mathtt{dec}(\mathtt{enc}(message, key), key) =_E message$.

Processes are constructed using the following grammar:

$P, Q, R :=$		plain processes
	0	null process
	$P \mid Q$	parallel composition
	$!P$	replication
	$\nu n.P$	name restriction ("new")
	$\mathtt{if}\ M = N\ \mathtt{then}\ P\ \mathtt{else}\ Q$	conditional
	$\mathtt{in}(u, x)$	message input
	$\mathtt{out}(u, x)$	message output

Active or extended processes are plain processes or active substitutions:

$A, B, C :=$		active processes
	P	plain process
	$A \mid B$	parallel composition
	$\nu n.A$	name restriction
	$\nu x.A$	variable restriction
	$\{^M/_x\}$	active substitution

The substitution $\{^M/_x\}$ replaces the variable x with term M. We denote $fv(A)$, $bv(A)$, $fn(A)$, $bn(A)$ the free variables, bound variables, free names or bound names respectively. A process is closed if all variables are bound or defined by an active substitution.

The *frame* $\Phi(A)$ of an extended process A is obtained when replacing all plain processes in A by 0. This frame can be seen as a representation of what is statically know to the exterior about a process. The domain $\mathtt{dom}(\Phi)$ of a frame Φ is the set of variables for which Φ defines a substitution. An evaluation context $C[\_]$ denotes an extended process with a hole for an extended process.

The semantics are described in Appendix A. We concentrate here on the equivalences needed to define our properties:

Definition 1 (Equivalence in a Frame)**.** *Two terms M and N are equal in the frame ϕ, written $(M = N)\phi$, if and only if $\phi \equiv \nu\tilde{n}.\sigma$, $M\sigma = N\sigma$, and $\{\tilde{n}\} \cap (fn(M) \cup fn(N)) = \emptyset$ for some names $\tilde{n}$ and some substitution σ.*

Definition 2 (Static Equivalence ($\approx_s$)). *Two closed frames ϕ and ψ are statically equivalent, written $\phi \approx_s \psi$, when dom(ϕ) =dom(ψ) and when for all terms M and N $(M = N)\phi$ if and only if $(M = N)\psi$.*

Two extended processes A and B are statically equivalent ($A \approx_s B$) if their frames are statically equivalent.

The intuition behind this definition is simple: Two processes are statically equivalent if the messages exchanged with the environment cannot be distinguished by an attacker (i.e. all operations on both sides give the same results). This idea can be extended to *labelled bisimilarity*.

Definition 3 (Labelled Bisimilarity ($\approx_l$)). *Labelled bisimilarity is the largest symmetric relation $\mathcal{R}$ on closed extended processes, such that $A\ \mathcal{R}\ B$ implies*

1. $A \approx_s B$,
2. *if $A \rightarrow A'$, then $B \rightarrow B'$ and $A'\ \mathcal{R}\ B'$ for some B',*
3. *if $A \xrightarrow{\alpha} A'$ and $fv(\alpha) \subseteq$ dom(A) and $bn(\alpha) \cap fn(B) = \emptyset$, then $B \rightarrow^* \xrightarrow{\alpha} \rightarrow^* B'$ and $A'\ \mathcal{R}\ B'$ for some B'.*

In this case each interaction on one side can be simulated by the other side, and the processes are statically equivalent at each step during the execution, thus an attacker cannot distinguish both sides. Labelled bisimilarity implies "classic" bisimilarity [1], but is often easier to prove and can be used to express many classical security properties, in particular anonymity properties.

2.2 Voting Process

We use the definition by Delaune et al. [7] to model voting protocols in the applied pi calculus. The basic idea is simple: A voting process is the parallel composition of all voters and the trusted authorities, whereas untrusted authorities are left to the context (i.e. the attacker). Messages are exchanged over public or private channels. We limit ourselves to protocols where each voter votes only once.

Definition 4 (Voting Process [7]). *A voting process is a closed plain process*

$$VP \equiv \nu\tilde{n}.(V\sigma_1|\ldots|V\sigma_n|A_1|\ldots|A_m).$$

The $V\sigma_i$ are the voter processes, the A_js the honest election authorities, and the $\tilde{n}$ are channel names. We also suppose that $v \in$ dom(σ_i) is a variable which refers to the value of the vote. We define an evaluation context S which is like VP, but has a hole instead of three $V\sigma_i$, and an evaluation context S' which is like VP, but has a hole instead of two $V\sigma_i$.

Note that S and S' contain – by construction – only honest voters, i.e. voters that follow the protocol and do not collude with the attacker.

2.3 Privacy

Before discussing Vote-Independence, we recall the definition of the three basic privacy properties as given by Delaune et al. [7].

2.3.1 Vote-Privacy

The intuition for Vote-Privacy is the following: An attacker cannot distinguish two runs of the voting protocols where two voters swap their votes. This does not change the outcome and if the votes are private, the attacker should not know which vote belongs to which voter:

Definition 5 (Vote-Privacy [7]). *A voting process respects* Vote-Privacy (P) *if for all votes a and b*

$$S'\left[V_A\left\{^a/_v\right\}|V_B\left\{^b/_v\right\}\right] \approx_l S'\left[V_A\left\{^b/_v\right\}|V_B\left\{^a/_v\right\}\right]$$

2.3.2 Receipt-Freeness

To define Receipt-Freeness, we use the transformation P^{ch} which can be applied to a process P. The transformed process outputs all its inputs and its private data (in particular new names, for example random values) on a special channel ch to the attacker. In the case of a voting process V, this corresponds to trying to create a receipt of the vote. If a protocol is receipt-free, a voter should be able to fake all these outputs to the coercer, i.e. to output fake values without the attacker noticing. This means that there should exist some process V' so that the attacker is not able to distinguish between a successfully coerced voter V^{ch} that votes c and outputs the correct values, and a voter V' that fakes the values and votes a instead. To ensure that the coercer cannot tell both cases apart from the result, Delaune et al. introduce another voter that counterbalances the vote, and require that V' actually votes for a using Definition 7.

Definition 6 (Process P^{ch} [7])**.** *Let P be a plain process and ch be a channel name. We define P^{ch} as follows:*

- $0^{ch} \mathrel{\hat{=}} 0$,
- $(P|Q)^{ch} \mathrel{\hat{=}} P^{ch}|Q^{ch}$,
- $(\nu n.P)^{ch} \mathrel{\hat{=}} \nu n.\mathtt{out}(ch, n).P^{ch}$ *when n is a name of base type,*
- $(\nu n.P)^{ch} \mathrel{\hat{=}} \nu n.P^{ch}$ *otherwise,*
- $(\mathtt{in}(u, x).P)^{ch} \mathrel{\hat{=}} \mathtt{in}(u, x).\mathtt{out}(ch, x).P^{ch}$ *when x is a variable of base type,*
- $(\mathtt{in}(u, x).P)^{ch} \mathrel{\hat{=}} \mathtt{in}(u, x).P^{ch}$ *otherwise,*
- $(\mathtt{out}(u, M).P)^{ch} \mathrel{\hat{=}} \mathtt{out}(u, M).P^{ch}$,
- $(!P)^{ch} \mathrel{\hat{=}} !P^{ch}$,
- $(\mathtt{if}\ M = N\ \mathtt{then}\ P\ \mathtt{else}\ Q)^{ch} \mathrel{\hat{=}} \mathtt{if}\ M = N\ \mathtt{then}\ P^{ch}\ \mathtt{else}\ Q^{ch}$.

In the remainder we assume $ch \notin fn(P) \cup bn(P)$ before applying the transformation.

Definition 7 (Process $A^{\setminus out(ch,\cdot)}$ [7])**.** *Let A be an extended process. We define the process $A^{\setminus out(ch,\cdot)}$ as $\nu ch.(A|!in(ch, x))$.*

Definition 8 (Receipt-Freeness [7])**.** *A voting process respects* Receipt-Freeness (RF) *if there exists a closed plain process V' such that for all votes a and c we have*

$$V'^{\setminus out(chc,\cdot)} \approx_l V_A\{^a/_v\}$$

and

$$S'\left[V_A\{^c/_v\}^{chc}|V_B\{^a/_v\}\right] \approx_l S'\left[V'|V_B\{^c/_v\}\right]$$

2.3.3 Coercion-Resistance

Similarly to Receipt-Freeness, Delaune et al. define a process that outputs all its inputs and secret values to the attacker. To express interactive coercion, it additionally waits for input from the context that tells it what to do before outputting values or branching (Definition 9).

Definition 9 (Process P^{c_1,c_2} [7])**.** *Let P be a plain process and c_1, c_2 be channel names. We define P^{c_1,c_2} as follows:*

- $0^{c_1,c_2} \mathrel{\hat{=}} 0$,
- $(P|Q)^{c_1,c_2} \mathrel{\hat{=}} P^{c_1,c_2}|Q^{c_1,c_2}$,
- $(\nu n.P)^{c_1,c_2} \mathrel{\hat{=}} \nu n.\mathtt{out}(c_1, n).P^{c_1,c_2}$ *when n is a name of base type,*

- $(\nu n.P)^{c_1,c_2} \hat{=} \nu n.P^{c_1,c_2}$ *otherwise,*
- $(\mathtt{in}(u,x).P)^{c_1,c_2} \hat{=} \mathtt{in}(u,x).\mathtt{out}(c_1,x).P^{c_1,c_2}$ *when x is a variable of base type and x is a fresh variable,*
- $(\mathtt{in}(u,x).P)^{c_1,c_2} \hat{=} \mathtt{in}(u,x).P^{c_1,c_2}$ *otherwise,*
- $(\mathtt{out}(u,M).P)^{c_1,c_2} \hat{=} \mathtt{in}(c_2,x).\mathtt{out}(u,x).P^{c_1,c_2}$,
- $(!P)^{c_1,c_2} \hat{=} !P^{c_1,c_2}$,
- $(\mathtt{if}\ M = N\ \mathtt{then}\ P\ \mathtt{else}\ Q)^{c_1,c_2} \hat{=} \mathtt{in}(c_2,x).\mathtt{if}\ x = \mathit{true}\ \mathtt{then}\ P^{c_1,c_2}$ $\mathtt{else}\ Q^{c_1,c_2}$ *where and x is a fresh variable and true is a constant.*

The definition then follows the same basic idea as for Receipt-Freeness: there exists a process V' that can interact with the attacker and fake all necessary messages without the attacker noticing. Yet one has to add some condition to ensure that the attacker cannot distinguish both sides of the bisimilarity simply based on the result by forcing the coerced voter to vote d, which would change the outcome. Tho enforce this, Delaune et al. use a context C that models the part of the attacker which interacts with V_A. The conditions on C ensure that the attacker actually forces the voter to vote c, and not d and thus make sure the vote is counterbalanced by V_B.

Definition 10 (Coercion-Resistance [7])**.** *A voting process respects* Coercion-Resistance (CR) *if there exists a closed plain process V' such that for any $C = \nu c_1.\nu c_2.(\_|P)$ satisfying $\tilde{n} \cap fn(C) = \emptyset$ and $S\left[C\left[V_A\left\{^{?}/_{v}\right\}^{c_1,c_2}\right]|V_B\left\{^{a}/_{v}\right\}\right] \approx_l S\left[V_A\left\{^{c}/_{v}\right\}^{chc}|V_B\left\{^{a}/_{v}\right\}\right]$ we have for all votes a and c*

$$C\left[V'\right]^{\backslash out(chc,\cdot)} \approx_l V_A\left\{^{a}/_{v}\right\}$$

and

$$S'\left[C\left[V_A\left\{^{?}/_{v}\right\}^{c_1,c_2}\right]|V_B\left\{^{a}/_{v}\right\}\right] \approx_l S'\left[C\left[V'\right]|V_B\left\{^{c}/_{v}\right\}\right]$$

Note that we write $\left\{^{?}/_{v}\right\}$ to represent the fact that the coerced voters vote does not depend on the substitution, but on the interaction with the context C.

3 Vote-Independence

In the previous privacy definitions the attacker has the role of an outside observer that tries to infer something about someone's vote. In the case of Coercion-Resistance or Receipt-Freeness he might communicate with the targeted voter, but he cannot necessarily vote himself or collude with other voters - unlike what would generally happen in real-world elections.

To address this shortcoming and obtain a more realistic model of the attacker's abilities, we introduce the notion of Vote-Independence for different levels of collaboration. The idea is to extend the existing definitions to the case where the attacker can vote himself and might try to relate his vote to the vote of a targeted voter to compromise privacy (for example copy it as in the attack by B. Smyth and V. Cortier [21]).

3.1 Vote-Independence (without Collaboration)

Definition 11 (Vote-Independence)**.** *A voting process respects* Vote-Independence (VI) *if for all votes a and b*

$$S\left[V_A\left\{^{a}/_{v}\right\}|V_B\left\{^{b}/_{v}\right\}|V_C^{c_1,c_2}\right] \approx_l S\left[V_A\left\{^{b}/_{v}\right\}|V_B\left\{^{a}/_{v}\right\}|V_C^{c_1,c_2}\right]$$

The intuition behind our definition is the following: We start from the definition of privacy, but add a voter under the control of the attacker in both cases. If an attacker can relate his vote to the vote of one of the voters (for example copy V_A's vote, i.e. vote for the same candidate), he will be able to distinguish

both sides as the result of the vote will be different. This is the most basic definition, as the attacker has only access to publicly available data. Subsequently we add the possibility of collaborating voters.

Smyth and Cortier [21] used a similar idea in a recent extension to their original paper. Contrary to our definition, they implicitly include corrupted voters in the context S (or S' resp.). We chose to make the corrupted voter explicit to be able to easily compare both notions.

3.2 Vote-Independence with Passive Collaboration

Definition 12 (Vote-Independence with Passive Collaboration)**.** *A voting process respects* Vote - Independence with Passive Collaboration (VI-PC) *if there exists a closed plain process V' such that for all votes a and c*

$$V'^{\backslash out(chc,\cdot)} \approx_l V_A\{^a/_v\}$$

and

$$S\left[V_A\{^c/_v\}^{chc} | V_B\{^a/_v\} | V_C^{c_1,c_2}\right] \approx_l S[V' | V_B\{^c/_v\} | V_C^{c_1,c_2}]$$

Vote-Independence with Passive Collaboration can be seen analogously to Receipt-Freeness. The attacker should not be able to link his vote to another voter's vote, even if this voter collaborates with him and gives him access to his secret values after voting (secret keys, random values, nonces, etc.). This is ensured in the definition as the attacker cannot decide if he is in a case where the attacked voter actually collaborates with him, or if the voter only pretends to collaborate and in reality votes differently. If he could use the information provided by the attacked voter to e.g. copy his vote, he would be able to distinguish these cases.

3.3 Vote-Independence with Active Collaboration

Definition 13 (Vote-Independence with Active Collaboration)**.** *A voting process respects* Vote - Independence with Active Collaboration (VI-AC) *if there exists a closed plain process V' such that for any $C = \nu c_1.\nu c_2.(\_|P)$ satisfying $\tilde{n} \cap fn(C) = \emptyset$ and*

$$S[C[V_A\{^?/_v\}^{c_1,c_2}] | V_B\{^a/_v\} | V_C^{c_3,c_4}] \approx_l S\left[V_A\{^c/_v\}^{chc} | V_B\{^a/_v\} | V_C^{c_3,c_4}\right]$$

and for all votes a and c we have

$$C[V']^{\backslash out(chc,\cdot)} \approx_l V_A\{^a/_v\}$$

and

$$S[C[V_A\{^?/_v\}^{c_1,c_2}] | V_B\{^a/_v\} | V_C^{c_3,c_4}] \approx_l S[C[V'] | V_B\{^c/_v\} | V_C^{c_3,c_4}]$$

In this definition, the attacker is even more powerful. Similarly to Coercion-Resistance, he can interact with the attacked voter during the entire voting process.

4 Hierarchy and Relation to Privacy

4.1 Hierarchy

Intuitively, $VI - AC$ is a stronger property than $VI - PC$, which is a stronger property than VI. The following proposition confirms this intuition:

Proposition 1. *We have:*

- *If a protocol respects Vote-Independence with Active Collaboration, it also respects Vote - Independence with Passive Collaboration.*
- *If a protocol respects Vote-Independence with Passive Collaboration, it also respects Vote - Independence (without collaboration).*

The detailed proofs can be found in Appendix B.1 and B.2.

4.2 Relation to Privacy

The only difference between P and VI (or $VI-PC$ and RF, or $VI-AC$ and CR) is the process $V_C^{c_1,c_2}$, i.e. the existence of a legitimate voter that is under control of the attacker. Intuitively this gives the attacker more power and thus VI (or $VI-PC$ or $VI-AC$) should be the stronger property. Indeed:

Proposition 2. *We have:*

- *If a protocol respects Vote-Independence, it also respects Vote-Privacy.*
- *If a protocol respects Vote-Independence with Passive Collaboration, it also respects Receipt - Freeness.*
- *If a protocol respects Vote-Independence with Active Collaboration, it also respects Coercion - Resistance.*

Informally we can argue that any attack on vote-privacy can be used to break vote independence. In this case the voter under control of the attacker simply behaves as a normal voter and the attacker can employ the same attack. The formal proof is given in Appendix C. $CR \Rightarrow RF \Rightarrow P$ has been shown in the literature [7].

4.3 The Global Picture

Taking these properties together, we obtain the following hierarchy of notions. $A \Rightarrow B$ means that any protocol ensuring property A also ensures property B.

$$\begin{array}{ccc} \text{CR [15]} & \begin{array}{c}\Leftarrow\\ \not\Rightarrow\end{array} & \text{VI-AC [4]} \\ \Downarrow\not\Uparrow & & \Downarrow\not\Uparrow \\ \text{RF} & \begin{array}{c}\Leftarrow\\ \not\Rightarrow\end{array} & \text{VI-PC [19]} \\ \Downarrow\not\Uparrow & & \Downarrow\not\Uparrow \\ \text{P} & \begin{array}{c}\Leftarrow\\ \not\Rightarrow\end{array} & \text{VI [10]} \end{array}$$

The cited protocols [4, 10, 15, 19] illustrate the hierarchy and show that the inverse implications are not true, as discussed below.

4.4 Example: FOO

The protocol by Fujioka et al. [10] is based on commitments and blind signatures. It was proven to respect Vote-Privacy (P) [7], but is not Receipt-Free (RF) as the randomness of the commitment can be used as a receipt. We show that it ensures Vote-Independence (VI).

4.4.1 Informal Description

The protocol is split in three phases. In the first phase, the voter obtains the administrator's signature on a commitment to his vote:

- Voter V_i chooses his vote v_i and computes a commitment $x_i = \xi(v_i, k_i)$ for a random key k_i.
- He blinds the commitment using a blinding function χ, a random value r_i and obtains $e_i = \chi(x_i, r_i)$.
- He signs e_i and sends the signature $s_i = \sigma_{V_i}(e_i)$ together with e_i and his identity to the administrator.
- The administrator checks if V_i has the right to vote, has not yet voted, and if the signature s_i is correct. If all tests succeed, he signs $d_i = \sigma_A(e_i)$ and sends it back to V_i.

- V_i unblinds the signature and obtains $y_i = \delta(d_i, r_i)$. He checks the signature.

In the second phase, the actual voting takes place:

- Voter V_i sends (x_i, y_i) to the collector C through an anonymous channel.
- The collector checks the administrator's signature and enters (x_i, y_i) into a list.

When all ballots are cast or when the deadline is over, the counting phase begins:

- The collector publishes the list of correct ballots.
- V_i verifies that his commitment appears on the list and sends r_i together with the commitment's index l on the list to C using an anonymous channel.
- The collector C opens the l-th ballot using r_i and publishes the vote.

4.4.2 Model in Applied Pi Calculus

Our model is based on the one developed in [7], but we add a third voter. We use the following equational theory:

$$\begin{aligned} \mathtt{open}(\mathtt{commit}(m,r),r) &= m \\ \mathtt{checksign}(\mathtt{sign}(m,sk),\mathtt{pk}(sk)) &= m \\ \mathtt{unblind}(\mathtt{blind}(m,r),r) &= m \\ \mathtt{unblind}(\mathtt{sign}(\mathtt{blind}(m,r),sk),r) &= \mathtt{sign}(m,sk) \end{aligned}$$

The complete model can be found in Appendix D.

4.4.3 Analysis

Proposition 3. *FOO respects Vote-Independence (VI).*

Proof. Similarly to the proof of Vote-Privacy by [7], we do not need to trust any authority except for the key distribution process (`processK`). Thus the voter $V_C^{c_1,c_2}$ under control of the attacker only interacts with the attacker (as untrusted authorities are left to the context, i.e. the attacker), except during the key distribution process at the beginning. In this process he obtains his key (which we do not require to be secret) and the administrator's public key, which is available to the attacker anyway. Thus the attacker is essentially in the same situation as in the proof of Vote-Privacy. The full proof can be found in Appendix D. □

Note that this protocol cannot respect Vote-Independence with Passive or Active Collaboration ($VI-PC$ or $VI-AC$), as this would imply Receipt-Freeness (see the hierarchy). This shows that $VI \not\Rightarrow VI-PC$.

4.5 Example: Okamoto

The protocol by Okamoto [19] uses trap-door commitments to achieve receipt-freeness [7]. However it is not Coercion-Resistant (CR) [7].

4.5.1 Informal Description

The protocol is very similar to the one by Fujioka et al. [10] discussed above. The only difference is the use of a trap-door commitment and a timeliness member to open the commitments. The first phase - during which the voter obtains a signature on his commitment - follows the same protocol, except for the fact that this time ξ is a trapdoor-commitment. In the second phase the actual voting takes place:

- Voter V_i sends the signed trap-door commitment to the collector C through an anonymous channel.
- The collector checks the administrator's signature and enters (x_i, y_i) into a list.

- The voter sends (v_i, r_i, x_i) to the timeliness member through an untappable anonymous channel

When all ballots are cast or when the deadline is over, the counting phase begins:

- The collector publishes the list of correct ballots.
- V_i verifies that his commitment appears on the list.
- The timeliness member publishes a randomly shuffled list of votes v_i and a zero-knowledge proof that he knows a permutation π for which $x_{\pi(i)} = \xi(v_i, r_i)$.

4.5.2 Model in Applied Pi Calculus

Our model is based on the model used in [7], but we add a third voter. It is based on the following equational theory:

$$\begin{aligned}
\mathtt{open}(\mathtt{tdcommit}(m, r, td), r) &= m \\
\mathtt{tdcommit}(m_1, r, td) &= \mathtt{tdcommit}(m_2, f(m_1, r, td, m_2), td) \\
\mathtt{checksign}(\mathtt{sign}(m, sk), \mathtt{pk}(sk)) &= m \\
\mathtt{unblind}(\mathtt{blind}(m, r), r) &= m \\
\mathtt{unblind}(\mathtt{sign}(\mathtt{blind}(m, r), sk), r) &= \mathtt{sign}(m, sk)
\end{aligned}$$

The first equation models the creation of a trap-door commitment to m using a random value r and a trap-door td, whereas the second equation allows the construction of another random value to open a commitment differently. This requires knowledge of the trap-door td and the initial random value r.

4.5.3 Analysis

Proposition 4. *The protocol by Okamoto respects $VI - PC$.*

Proof. To prove this, we need to find a process V' that successfully fakes all secrets to a coercer. In addition to normal receipt-freeness, we also have to ensure that the attacker cannot use the secrets to e.g. copy the vote of the targeted voter.

In this protocol the trap-door commitment allows the voter to return a faked random number to the attacker which opens the commitment to any value the voter wants. This means that - although the attacker has access to the commitment and the necessary values to open it - he will always open it in a way that yields a vote for c due to the fake randomness, even if the voter actually committed to a. The same reasoning applies for copying votes: Although it is technically possible to copy the vote of the targeted voter, the voter will provide a faked random value, which will make the timeliness member open the vote as a vote for c. This makes it impossible for the attacker to know if the voter complied with his instructions or only pretended to do so, even if he tries to relate his vote to the targeted voter's vote.

The detailed model and complete proof can be found in Appendix E. □

Thus the protocol also respects simple Vote-Independence (VI). Note that this protocol cannot respect Vote-Independence with Active Collaboration ($VI - AC$), as this would imply Coercion-Resistance. This shows that $VI - PC \not\Rightarrow VI - AC$.

4.6 Example: Bingo Voting

Bingo Voting was developed by Bohli et al. [4] to achieve coercion-resistance as well as individual and universal verifiability by using a trusted random number generator (RNG). We use Bingo Voting to illustrate the existence of protocols that respect Vote-Independence with active Collaboration ($VI - AC$).

4.6.1 Informal Description

We consider an election with k voters and l candidates. The protocol is split into three phases: The pre-voting phase, the voting phase and the post-voting phase. In the pre-voting phase, the voting machine generates k random values $n_{i,j}$ for every candidate p_j (the dummy votes). It commits to the $k \cdot l$ pairs $(n_{i,j}, p_j)$ and publishes the shuffled commitments.

In the voting phase, the voter enters the voting booth and selects the candidate he wants to vote for on the voting machine. The RNG generates a random number r, which is transmitted to the voting machine and displayed to the voter. The voting machine chooses for each candidate a dummy vote, except for the voter's choice. For this candidate, the random value from the RNG is used and the receipt (a list of all candidates and the corresponding random numbers) is created. Finally, the voter checks that the number displayed on the RNG corresponds to the entry of his candidate on the receipt.

In the post-voting phase, the voting machine announces the result, publishes all receipts and opens the commitments of all unused dummy votes. The machine also generates non-interactive zero-knowledge proofs that each unopened commitment was actually used as a dummy vote in one of the receipts.

4.6.2 Model in Applied Pi Calculus

As we are only interested in privacy, we ignore the zero-knowledge proofs which are necessary to achieve verifiability. This yields a very simple equational theory:

$$\mathtt{open}(\mathtt{commit}(m, r), r) \quad = \quad m$$

We assume the voting machine to be honest, otherwise no privacy can be guaranteed as the vote is submitted in clear by the voter. The detailed model can be found in Appendix F.

4.6.3 Analysis

Proposition 5. *Bingo Voting respects $VI - AC$.*

Proof. The receipts contain only random values which makes it impossible for the attacker to know if a certain number corresponds to the random value by the RNG or a dummy vote. Thus the voter V' does not even have to fake a receipt, he can simply forward his receipt and claim he voted for the coercer's choice. Constructing a related vote based on the receipt is not possible either since - while voting - the attacker has to transmit his choice in clear to the voting machine. Being able to e.g. copy V_A's vote would imply the break of simple privacy on the voter's vote using the receipt.

The complete proof can be found in Appendix F □

This implies that Bingo Voting is coercion resistant and provides Vote-Independence.

4.7 Example: Lee et al.

The protocol by Lee et al. [15] was proven to be Coercion-Resistant (CR) in [7], but does not respect Vote-Independence (VI) – and thus neither $VI - PC$ nor $VI - AC$ – as we show. It is based on trusted devices that re-encrypt ballots and use designated verifier proofs (DVPs) to prove their correct behavior to the voter.

4.7.1 Informal Description

We simplified the protocol to focus on the important parts with respect to privacy and vote-independence. For example, we do not consider distributed authorities.

- The administrator sets up the election, distributes keys and registers legitimate voters. Each voter is equipped with his personal trusted device. At the end, he publishes a list of legitimate voters and corresponding trusted devices.

- The voter encrypts his vote with the tallier's public key (using the El Gamal scheme), signs it and sends it to his trusted device over a private channel. The trusted device verifies the signature, re-encrypts and signs the vote, and returns it, together with a DVP that the re-encryption is correct, to the voter. The voter verifies the signature and the proof, double signs the ballot and publishes it on the bulletin board.
- The administrator verifies for all ballots if the voter has the right to vote and if the vote is correctly signed. He publishes the list of correct ballots, which is then shuffled by the mixer.
- The tallier decrypts the mixed votes and publishes the result.

4.7.2 Model in Applied Pi Calculus

Our model is based on the one developed in [7], but we add a third (corrupted) voter and an explicit mixing stage. This stage was left out in their model, but is essential to highlight the difference between Vote-Privacy and Vote-Independence. We use the following equational theory:

$$\begin{gathered}
\mathtt{decrypt}(\mathtt{penc}(m, \mathtt{pk}(sk), r), sk) = m \\
\mathtt{checksign}(\mathtt{sign}(m, sk), \mathtt{pk}(sk)) = m \\
\mathtt{rencrypt}(\mathtt{penc}(m, \mathtt{pk}(sk), r1), r2) = \mathtt{penc}(m, \mathtt{pk}(sk), f(r1, r2)) \\
\mathtt{checkdvp}(\mathtt{dvp}(x, \mathtt{rencrypt}(x, r), r, \mathtt{pk}(sk)), x, \mathtt{rencrypt}(x, r), \mathtt{pk}(sk)) = \mathtt{ok} \\
\mathtt{checkdvp}(\mathtt{dvp}(x, y, z, skv), x, y, \mathtt{pk}(skv)) = \mathtt{ok}
\end{gathered}$$

4.7.3 Analysis

In the extended model, the protocol by Lee et al. still ensures (CR), but it is not (VI).

Proposition 6. *The protocol by Lee et al. does not respect Vote-Independence (VI).*

Proof. As acknowledged by the authors in their original paper [15], it is possible to copy votes. More precisely, an attacker can access the ballots on the bulletin board before the mixing takes place. He can easily verify which ballot belongs to which voter as they are signed by the voters themselves. He can remove the signature and use the ciphertext as an input to his trusted device. The trusted device will re-encrypt and sign it. This allows the attacker to construct a correct ballot which contains the same vote as the targeted honest voter. This obviously contradicts vote-independence.

Appendix G shows how this can be seen in our model. □

This example shows that vote-independence properties are strictly stronger than the corresponding privacy properties ($CR \not\Rightarrow VI - AC$, $RF \not\Rightarrow VI - PC$, $P \not\Rightarrow VI$), as even a coercion-resistant protocol fails to respect simple vote-independence.

5 Conclusion

Inspired by an attack based on copying votes, we extended the classical threat model and developed the notion of "Vote-Independence". We gave a formal definition and showed that it is stronger than standard vote-privacy. We generalized the definition to passive and active collaboration, and obtained refined properties on the same attack level as receipt-freeness and coercion-resistance.

Subsequently we analyzed practical examples which illustrate that our property is strictly stronger, i.e. that even coercion resistant protocols can fail with respect to Vote-Independence, and thus of practical interest.

Future Work. We plan to translate our symbolic definition to the computational model and extend our analysis e.g. to accommodate protocols permitting multiple votes. Additionally, it would be desirable to develop tools that at least partly automate and/or verify the necessary proofs.

References

[1] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '01, pages 104–115, New York, 2001. ACM. 2.1, 2.1

[2] Ben Adida, Olivier De Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. Electing a university president using open-audit voting: analysis of real-world use of helios. In *Proceedings of the 2009 conference on Electronic voting technology/workshop on trustworthy elections*, EVT/WOTE'09, pages 10–10, Berkeley, CA, USA, 2009. USENIX Association. 1

[3] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, February–March 2008. 2.1

[4] Jens-Matthias Bohli, Jörn Müller-Quade, and Stefan Röhrich. Bingo voting: Secure and coercion-free voting using a trusted random number generator. In Ammar Alkassar and Melanie Volkamer, editors, *E-Voting and Identity*, volume 4896 of *Lecture Notes in Computer Science*, pages 111–124. Springer Berlin / Heidelberg, 2007. 1, 4.3, 4.6

[5] UK Electoral Commission. Key issues and conclusions: May 2007 electoral pilot schemes. http://www.electoralcommission.org.uk/elections/pilots/May2007. 1

[6] Bundesverfassungsgericht (Germany's Federal Constitutional Court). Use of voting computers in 2005 bundestag election unconstitutional, March 2009. Press release 19/2009 http://www.bundesverfassungsgericht.de/en/press/bvg09-019en.html. 1

[7] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17:435–487, December 2009. 1, 1, 2.2, 4, 2.3, 5, 6, 7, 8, 9, 10, 4.2, 4.4, 4.4.2, 4.4.3, 4.5, 4.5.2, 4.7, 4.7.2, A, B.1, B.2, 2, B.2, D

[8] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Verifying privacy-type properties of electronic voting protocols: A taster. In David Chaum, Markus Jakobsson, Ronald L. Rivest, Peter Y. A. Ryan, Josh Benaloh, Mirosław Kutyłowski, and Ben Adida, editors, *Towards Trustworthy Elections – New Directions in Electronic Voting*, volume 6000 of *Lecture Notes in Computer Science*, pages 289–309. Springer, May 2010. 1

[9] Jannik Dreier, Pascal Lafourcade, and Yassine Lakhnech. Vote-independence: A powerful privacy notion for voting protocols. In *Proceedings of the 4th Workshop on Foundations & Practice of Security (FPS)*, Lecture Notes in Computer Science. Springer, 2011. 1

[10] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In Jennifer Seberry and Yuliang Zheng, editors, *Advances in Cryptology – AUSCRYPT '92*, volume 718 of *Lecture Notes in Computer Science*, pages 244–251. Springer Berlin / Heidelberg, 1992. 1, 4.3, 4.4, 4.5.1

[11] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections, 2002. Cryptology ePrint Archive, Report 2002/165, http://eprint.iacr.org/. 1

[12] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, WPES '05, pages 61–70, New York, NY, USA, 2005. ACM. 1

[13] Steve Kremer and Mark Ryan. Analysis of an electronic voting protocol in the applied pi calculus. In *Proceedings of the 14th European Symposium On Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 186–200. Springer, 2005. 1

[14] Steve Kremer, Mark Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *Proceedings of the 15th European Symposium on Research in Computer Security, ESORICS 2010*, volume 6345 of *Lecture Notes in Computer Science*, pages 389–404. Springer, 2010. 1

[15] Byoungcheon Lee, Colin Boyd, Ed Dawson, Kwangjo Kim, Jeongmo Yang, and Seungjae Yoo. Providing receipt-freeness in mixnet-based voting protocols. In Jong In Lim and Dong Hoon Lee, editors, *Information Security and Cryptology - ICISC 2003*, volume 2971 of *Lecture Notes in Computer Science*, pages 245–258. Springer Berlin / Heidelberg, 2004. 1, 4.3, 4.7, 4.7.3

[16] Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 373–392. Springer, 2006. 1

[17] Aybek Mukhamedov and Mark D. Ryan. Identity escrow protocol and anonymity analysis in the applied pi-calculus. *ACM Transactions on Information System Security*, 13(41):1–29, December 2010. F

[18] Participants of the Dagstuhl Conference on Frontiers of E-Voting. Dagstuhl accord, 2007. http://www.dagstuhlaccord.org/. 1

[19] Tatsuaki Okamoto. An electronic voting scheme. In *Proceedings of the IFIP World Conference on IT Tools*, pages 21–30, 1996. 1, 4.3, 4.5

[20] Ben Smyth and Veronique Cortier. Attacking and fixing helios: An analysis of ballot secrecy. Accepted at CSF'11. 1

[21] Ben Smyth and Veronique Cortier. Attacking and fixing helios: An analysis of ballot secrecy. Cryptology ePrint Archive, Report 2010/625, 2010. http://eprint.iacr.org/. (document), 1, 3, 3.1

[22] Ben Smyth, Mark D. Ryan, Steve Kremer, and Mounira Kourjieh. Towards automatic analysis of election verifiability properties. In Alessandro Armando and Gavin Lowe, editors, *Proceedings of the Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (ARSPA-WITS'10)*, volume 6186 of *Lecture Notes in Computer Science*, pages 146–163, Paphos, Cyprus, October 2010. Springer. 1

[23] Ministerie van Binnenlandse Zaken en Koninkrijksrelaties (Netherland's Ministry of the Interior and Kingdom Relations). Stemmen met potlood en papier (voting with pencil andpaper), May 2008. Press release http://www.minbzk.nl/onderwerpen/grondwet-en/verkiezingen/nieuws--en/112441/stemmen-met-potlood. 1

A Applied Pi Calculus

The semantics of the calculus are given by *Structural equivalence* ($\equiv$), which is defined as the smallest equivalence relation on extended processes that is closed under application of evaluation contexts, α-conversion on names and variables such that:

PAR-0	$A\|0 \equiv A$	REPL	$!P \equiv P\|!P$
PAR-A	$A\|(B\|C) \equiv (A\|B)\|C$	REWRITE	$\{^M/_x\} \equiv \{^N/_x\}$
PAR-C	$A\|B \equiv B\|A$		if $M =_E N$
NEW-0	$\nu n.0 \equiv 0$	ALIAS	$\nu x.\{^M/_x\} \equiv 0$
NEW-C	$\nu u.\nu v.A \equiv \nu v.\nu u.A$	SUBST	$\{^M/_x\}\|A \equiv \{^M/_x\}\|A\{^M/_x\}$
	NEW-PAR $A\|\nu u.B \equiv \nu u.(A\|B)$ if $u \notin fn(A) \cup fn(b)$		

and extended by *Internal reduction* ($\rightarrow$), the smallest relation on extended processes closed by structural equivalence and application of evaluation contexts such that:

$$\begin{array}{lrcl}
\text{COMM} & \mathtt{out}(a,x).P \mid \mathtt{in}(a,x).Q & \to & P \mid Q \\
\text{THEN} & \mathtt{if}\ M = M\ \mathtt{then}\ P\ \mathtt{else}\ Q & \to & P \\
\text{ELSE} & \mathtt{if}\ M = N\ \mathtt{then}\ P\ \mathtt{else}\ Q & \to & Q \\
 & \text{for any ground terms such that } M \neq_E N & &
\end{array}$$

To describe the interaction of processes with the exterior, we use labeled operational semantics ($\xrightarrow{\alpha}$) where α can be an input or the output of a channel name or a variable of base type:

$$\begin{array}{ll}
\text{IN} & \mathtt{in}(a,x).P \xrightarrow{\mathtt{in}(a,M)} P\{^M/_x\} \\
\text{OUT-ATOM} & \mathtt{out}(a,u).P \xrightarrow{\mathtt{out}(a,u)} P \\
\text{OPEN-ATOM} & \dfrac{A \xrightarrow{\mathtt{out}(a,u)} A' \qquad u \neq a}{\nu u.A \xrightarrow{\nu u.\mathtt{out}(a,u)} A'} \\
\text{SCOPE} & \dfrac{A \xrightarrow{\alpha} A' \qquad u \text{ does not occur in } \alpha}{\nu u.A \xrightarrow{\alpha} \nu u.A'} \\
\text{PAR} & \dfrac{A \xrightarrow{\alpha} A' \qquad bv(\alpha) \cap fv(B) = bn(\alpha) \cap fn(B) = \emptyset}{A \mid B \xrightarrow{\alpha} A' \mid B} \\
\text{STRUCT} & \dfrac{A \equiv B \qquad B \xrightarrow{\alpha} B' \qquad B' \equiv A'}{A \xrightarrow{\alpha} A'}
\end{array}$$

Labeled transitions are not closed under the evaluation contexts. Note that a term M cannot be output directly, but only a variable as "reference" to it. This is to model that e.g. the output of $\mathtt{enc}(m,k)$ does not give the context access to m.

In our proofs we will make use of the following Lemma 1 by [7]:

Lemma 1. *Let $C_1 = \nu\tilde{u}_1.(\_|B_1)$ and $C_2 = \nu\tilde{u}_2.(\_|B_2)$ be two evaluation contexts such that $\tilde{u}_1 \cap (fv(B_2) \cup fn(B_2)) = \emptyset$ and $\tilde{u}_2 \cap (fv(B_1) \cup fn(B_1)) = \emptyset$. Then we have $C_1[C_2[A]] \equiv C_2[C_1[A]]$ for any extended process A.*

B Proof of Propostion 1

B.1 "$VI - AC \Rightarrow VI - PC$"

The proof is very similar to the proof of $CR \Rightarrow RF$ in [7].

Proof. Let C be an evaluation context such that $C = \nu c_1.\nu c_2.(\_|P)$ for some plain process P which fulfills

$$S\left[C\left[V_A\left\{^?/_v\right\}^{c_1,c_2}\right]|V_B\left\{^a/_v\right\}|V_C^{c_3,c_4}\right] \approx_l S\left[V_A\left\{^c/_v\right\}^{chc}|V_B\left\{^a/_v\right\}|V_C^{c_3,c_4}\right]$$

Note that such a C can be constructed directly from the vote process V. By hypothesis we know that there is a closed plain process V' so that

$$C\left[V'\right]^{\setminus out(chc,\cdot)} \approx_l V_A\left\{^a/_v\right\}$$

and

$$S\left[C\left[V_A\left\{^?/_v\right\}^{c_1,c_2}\right]|V_B\left\{^a/_v\right\}|V_C^{c_3,c_4}\right] \approx_l S\left[C\left[V'\right]|V_B\left\{^c/_v\right\}|V_C^{c_3,c_4}\right]$$

We have to find another process V'' so that

$$V''^{\setminus out(chc,\cdot)} \approx_l V_A\left\{^a/_v\right\}$$

and

$$S\left[V_A\left\{^c/_v\right\}^{chc}|V_B\left\{^a/_v\right\}|V_C^{c_3,c_4}\right] \approx_l S\left[V''|V_B\left\{^c/_v\right\}|V_C^{c_3,c_4}\right]$$

Let $V'' = C[V']$. This directly fulfills the first requirement. We now use the hypotheses

$$S\left[C\left[V_A\left\{^?/_v\right\}^{c_1,c_2}\right]|V_B\left\{^a/_v\right\}|V_C^{c_3,c_4}\right] \approx_l S\left[V_A\left\{^c/_v\right\}^{chc}|V_B\left\{^a/_v\right\}|V_C^{c_3,c_4}\right]$$

and

$$S\left[C\left[V_A\left\{^{?}/_{v}\right\}^{c_1,c_2}\right]|V_B\left\{^{a}/_{v}\right\}|V_C^{c_3,c_4}\right] \approx_l S\left[C\left[V'\right]|V_B\left\{^{c}/_{v}\right\}|V_C^{c_3,c_4}\right]$$

As labelled bisimilarity is transitive, we can conclude

$$S\left[V_A\left\{^{c}/_{v}\right\}^{chc}|V_B\left\{^{a}/_{v}\right\}|V_C^{c_3,c_4}\right] \approx_l S\left[C\left[V'\right]|V_B\left\{^{c}/_{v}\right\}|V_C^{c_3,c_4}\right]$$

which gives us the desired result for $V'' = C[V']$. □

B.2 "$VI - PC \Rightarrow VI$"

For this proof, we need the following Lemma by [7].

Lemma 2 ([7]). *Let P be a closed plain process and ch a channel name such that $ch \notin fn(P) \cup bn(P)$. We have $(P^{ch})^{\setminus out(ch,\cdot)} \approx_l P$.*

The proof itself is similar to the proof of $RF \Rightarrow P$ in [7]:

Proof. By hypothesis there is a closed plain process so that

$$V'^{\setminus out(chc,\cdot)} \approx_l V_A\left\{^{a}/_{v}\right\}$$

and

$$S\left[V_A\left\{^{c}/_{v}\right\}^{chc}|V_B\left\{^{a}/_{v}\right\}|V_C^{c_1,c_2}\right] \approx_l S\left[V'|V_B\left\{^{c}/_{v}\right\}|V_C^{c_1,c_2}\right]$$

We apply the context $\nu chc(\_|!\mathtt{in}(chc,x))$ on both sides, which gives

$$S\left[V_A\left\{^{c}/_{v}\right\}^{chc}|V_B\left\{^{a}/_{v}\right\}|V_C^{c_1,c_2}\right]^{\setminus out(chc,\cdot)} \approx_l S\left[V'|V_B\left\{^{c}/_{v}\right\}|V_C^{c_1,c_2}\right]^{\setminus out(chc,\cdot)}$$

By using Lemma 1 we obtain

$$S\left[V'|V_B\left\{^{c}/_{v}\right\}|V_C^{c_1,c_2}\right]^{\setminus out(chc,\cdot)} \equiv S\left[V'^{\setminus out(chc,\cdot)}|V_B\left\{^{c}/_{v}\right\}|V_C^{c_1,c_2}\right]$$

and

$$S\left[V_A\left\{^{c}/_{v}\right\}^{chc}|V_B\left\{^{a}/_{v}\right\}|V_C^{c_1,c_2}\right]^{\setminus out(chc,\cdot)} \equiv S\left[\left(V_A\left\{^{c}/_{v}\right\}^{chc}\right)^{\setminus out(chc,\cdot)}|V_B\left\{^{a}/_{v}\right\}|V_C^{c_1,c_2}\right]$$

We can now apply Lemma 2 and use the fact that labelled bisimilarity is closed under structural equivalence and obtain

$$S\left[V'^{\setminus out(chc,\cdot)}|V_B\left\{^{c}/_{v}\right\}|V_C^{c_1,c_2}\right] \approx_l S\left[V_A\left\{^{c}/_{v}\right\}|V_B\left\{^{a}/_{v}\right\}|V_C^{c_1,c_2}\right]$$

where we can use $V'^{\setminus out(chc,\cdot)} \approx_l V_A\left\{^{a}/_{v}\right\}$ to conclude. □

C Proof of Proposition 2 ("$VI \Rightarrow P$")

Proof. We will use a proof by contradiction. Suppose

$$S'\left[V_A\left\{^{a}/_{v}\right\}|V_B\left\{^{b}/_{v}\right\}\right] \not\approx_l S'\left[V_A\left\{^{b}/_{v}\right\}|V_B\left\{^{a}/_{v}\right\}\right]$$

which we can rewrite as

$$S\left[V_A\left\{^{a}/_{v}\right\}|V_B\left\{^{b}/_{v}\right\}|V_i\sigma_i\right] \not\approx_l S\left[V_A\left\{^{b}/_{v}\right\}|V_B\left\{^{a}/_{v}\right\}|V_i\sigma_i\right]$$

for some honest voter $V_i\sigma_i$[1]. This yields the contradiction for a context (an attacker) enforcing $V_C^{c_1,c_2} \approx_l V_i\sigma_i$. ↯ □

The proofs $VI - PC \Rightarrow RF$ and $VI - AC \Rightarrow CR$ are similar.

[1] Technically here we assume the existence of at least one other honest voter. This is unavoidable as we need at least three legitimate voters to be able to apply the definition of Vote-Independence.

```
(* private channels *)
ν privCh . ν pkaCh1 . ν pkaCh2 . ν skaCh.
ν skvaCh . ν skvbCh . ν skvcCh .
(* administrators *)
(processK | processA | processA | processA |
processC | processC | processC |
(* voters *)
(let skvCh = skvaCh in let v = a in processV) |
(let skvCh = skvbCh in let v = b in processV) |
(let skvCh = skvcCh in let v = c in processV))
```

Process 1: The main process

```
processK =
        (* private keys *)
        ν ska . ν skva . ν skvb . ν skvc.
        (* public keys *)
        let (pka, pkva, pkvb, pkvc)
        = (pk(ska), pk(skva), pk(skvb), pk(skvc)) in
        (* public key disclosure *)
        out(ch, pka).
        out(ch, pkva).out(ch, pkvb).out(ch, pkvc).
        (* register legitimate voters *)
        (out(privCh, pkva) | out(privCh, pkvb) |
         out(privCh, pkvc) | out(pkaCh1, pka) |
         out(pkaCh1, pka) | out(pkaCh1, pka) |
         out(pkaCh2, pka) | out(pkaCh2, pka) |
         out(pkaCh2, pka) | out(skaCh, ska) |
         out(skvaCh, skva) | out(skvbCh, skvb) |
         out(skvcCh, skvc))
```

Process 2: The key distribution process

D Proof of Proposition 3 ("FOO $\in VI$")

We use the following model:

The main process The main process (Process 1) sets up the private channels and executes the participation processes (three voters, three administrators, three collectors - one for each voter - and the keying process) in parallel.

The keying process The keying process (Process 2) creates the private keys, distributes them over private channels and publishes the corresponding public keys. He also sends the administrators a list containing the public keys of all legitimate voters.

The voting process The voter's process (Process 5) receives his private and the administrator's public key; then he votes following the protocol described informally in Section 4.4.

A voter under control of the attacker A voter controlled by the attacker is modelled by Process 6. This process is obtained when calculating $\texttt{processV}^{c_1,c_2}$ as defined in Definition 9.

```
processA =
        (* private key *)
        in(skaCh, ska).
        (* register legitimate voters *)
        in(privCh, pubkv).
        in(ch1, m1).
        let (pubkeyv, sig) = m1 in
        if pubkeyv = pubkv then
        out(ch2, sign(checksign(sig, pubkv), ska))
```

Process 3: The administrator process

```
processC =
        (* administrator's public key *)
        in(pkaCh2, pka).
        synch 1.
        in(ch3, (m3, m4)).
        if checksign(m4, pka) = m3 then
        ν l.
        out(ch4, (l, m3, m4)).
        in(ch5, (=l, rand)).
        let voteV = open(m3, rand) in
        out(ch, voteV)
```

Process 4: The collector process

```
processV =
        (* private key *)
        in(skvCh, skv).
        (* public key of administrator *)
        in(pkaCh1, pka).
        ν blinder . ν r .
        let commitedvote = commit(v, r) in
        let blindedvote = blind(commitedvote, blinder) in
        out(ch1, (pk(skv), sign(blindedvote, skv))).
        in(ch2, m2).
        let result = checksign(m2, pubka) in
        if result = blindedvote then
        let signedvote = unblind(m2, blinder) in
        synch 1.
        out(ch3, (commitedvote, signedvote)).
        in(ch4, (l, =commitedvote, =signedvote)).
        synch 2.
        out(ch5, (l, r))
```

Process 5: The voting process

```
processVc1c2 =
        (* private key *)
        in(skvCh, skv).out(c1, skv).
        (* public key of administrator *)
        in(pkaCh1, pka).out(c1, pka).
        ν blinder .out(c1, blinder). ν r .out(c1, r).
        let commitedvote = commit(v, r) in
        let blindedvote = blind(commitedvote, blinder) in
        in(c2, y1).out(ch1, y1).
        in(ch2, m2).out(c1, m2).
        let result = checksign(m2, pubka) in
        in(c2, y2).
        if y2 = true then
        let signedvote = unblind(m2, blinder) in
        synch 1.
        in(c2, y3).out(ch3, y3).
        in(ch4, m3).out(c1, m3).
        let (l, commitedvoter, signedvoter) = m3 in
        in(c2, y4).if y4 = true then
        in(c2, y5).if y4 = true then
        synch 2.
        in(c2, y6).out(ch5, y6)
```

Process 6: The voting process under control of the attacker

The administrator The administrator (Process 3) receives his private key and the public key of a legitimate voter. When receives the blinded commitment, he checks the signature, signs, and sends the result back.

The collector The collector (Process 4) receives the administrator's public key, which he then uses to verify the signature on incoming commitments. If the signature is correct, he creates a new bounded name l (the number in the list) and sends it together with the signed commitment back to the voter. The voter then reveals his randomness, which the collector uses to open the commitment.

Proof. Similar to [7], we will show that

$$\begin{array}{c} \nu pkaCh1.(V_A\{^a/_v\}\,|V_B\{^b/_v\}\,|V_C^{c_1,c_2}|\texttt{processK}) \\ \approx_l \\ \nu pkaCh1.(V_A\{^b/_v\}\,|V_B\{^a/_v\}\,|V_C^{c_1,c_2}|\texttt{processK}) \end{array} \quad (1)$$

where $V_A = \texttt{processV}\{^{skvaCh}/_{skvCh}\}$, $V_B = \texttt{processV}\{^{skvbCh}/_{skvCh}\}$ and $V_C^{c_1,c_2} = \texttt{processVc1c2}\{^{skvcCh}/_{skvCh}\}$. Note that this proof is technically only valid for two honest voters and one voter under control of the attacker. Nevertheless a similar proof can be made for other numbers of honest voters.

We will call the left hand side process P and the right hand side process Q. Both sides start by distributing the keys. The public key of the administrator is passed over a private channel which yields an internal reduction. We will not detail this part for better readability. $V_C^{c_1,c_2}$ will output the keys he receives, forward the attacker's input to the administrator and resend the answer to the administrator[2]. The honest

[2] As we do not trust the administrator, he is part of the context and thus of the attacker. This means that the attacker actually exchanges messages with himself.

voters construct the blinded and committed votes and send them to the administrator.

$$
\begin{aligned}
P \xrightarrow{\mathtt{in}(skvaCh,skva)} P_1 \xrightarrow{\mathtt{in}(skvbCh,skvb)} P_2 \xrightarrow{\mathtt{in}(skvcCh,skvc)} P_3 \rightarrow^* \\
\xrightarrow{\mathtt{out}(c1,x_1)} P_4 | \{^{skvc}/_{x_1}\} \xrightarrow{\mathtt{out}(c1,x_2)} P_5 | \{^{skvc}/_{x_1}\} | \{^{pka}/_{x_2}\} \\
\xrightarrow{\mathtt{in}(c2,y1)} P_6 | \{^{skvc}/_{x_1}\} | \{^{pka}/_{x_2}\} \\
\xrightarrow{\mathtt{out}(ch2,x_3)} P_7 | \{^{skvc}/_{x_1}\} | \{^{pka}/_{x_2}\} | \{^{y1}/_{x_3}\} \\
\xrightarrow{\mathtt{in}(ch2,m2)} P_8 | \{^{skvc}/_{x_1}\} | \{^{pka}/_{x_2}\} | \{^{y1}/_{x_3}\} \\
\xrightarrow{\mathtt{out}(c1,x_4)} P_9 | \{^{skvc}/_{x_1}\} | \{^{pka}/_{x_2}\} | \{^{y1}/_{x_3}\} | \{^{m2}/_{x_4}\} \\
\xrightarrow{\mathtt{out}(ch1,x_5)} \nu b_A.\nu r_A.(P_{10} | \{^{skvc}/_{x_1}\} | \{^{pka}/_{x_2}\} | \{^{y1}/_{x_3}\} | \{^{m2}/_{x_4}\} | \\
\{^{(pk(skva),sign(blind(commit(a,r_A),b_A),skva))}/_{x_5}\}) \\
\xrightarrow{\mathtt{out}(ch1,x_6)} \nu b_B.\nu b_A.\nu r_A.\nu r_B.(P_{11} | \{^{skvc}/_{x_1}\} | \{^{pka}/_{x_2}\} | \{^{y1}/_{x_3}\} | \\
\{^{m2}/_{x_4}\} | \{^{(pk(skva),sign(blind(commit(a,r_A),b_A),skva))}/_{x_5}\} | \\
\{^{(pk(skvb),sign(blind(commit(b,r_B),b_B),skvb))}/_{x_6}\})
\end{aligned}
\tag{2}
$$

Similarly

$$
\begin{aligned}
Q \xrightarrow{\mathtt{in}(skvaCh,skva)} Q_1 \xrightarrow{\mathtt{in}(skvbCh,skvb)} Q_2 \xrightarrow{\mathtt{in}(skvcCh,skvc)} Q_3 \rightarrow^* \\
\xrightarrow{\mathtt{out}(c1,x_1)} Q_4 | \{^{skvc}/_{x_1}\} \xrightarrow{\mathtt{out}(c1,x_2)} Q_5 | \{^{skvc}/_{x_1}\} | \{^{pka}/_{x_2}\} \\
\xrightarrow{\mathtt{in}(c2,y1)} Q_6 | \{^{skvc}/_{x_1}\} | \{^{pka}/_{x_2}\} \\
\xrightarrow{\mathtt{out}(ch2,x_3)} Q_7 | \{^{skvc}/_{x_1}\} | \{^{pka}/_{x_2}\} | \{^{y1}/_{x_3}\} \\
\xrightarrow{\mathtt{in}(ch2,m2)} Q_8 | \{^{skvc}/_{x_1}\} | \{^{pka}/_{x_2}\} | \{^{y1}/_{x_3}\} \\
\xrightarrow{\mathtt{out}(c1,x_4)} Q_9 | \{^{skvc}/_{x_1}\} | \{^{pka}/_{x_2}\} | \{^{y1}/_{x_3}\} | \{^{m2}/_{x_4}\} \\
\xrightarrow{\mathtt{out}(ch1,x_5)} \nu b_A.\nu r_A.(Q_{10} | \{^{skvc}/_{x_1}\} | \{^{pka}/_{x_2}\} | \{^{y1}/_{x_3}\} | \{^{m2}/_{x_4}\} | \\
\{^{(pk(skva),sign(blind(commit(b,r_A),b_A),skva))}/_{x_5}\}) \\
\xrightarrow{\mathtt{out}(ch1,x_6)} \nu b_B.\nu b_A.\nu r_A.\nu r_B.(Q_{11} | \{^{skvc}/_{x_1}\} | \{^{pka}/_{x_2}\} | \{^{y1}/_{x_3}\} | \\
\{^{m2}/_{x_4}\} | \{^{(pk(skva),sign(blind(commit(b,r_A),b_A),skva))}/_{x_5}\} | \\
\{^{(pk(skvb),sign(blind(commit(a,r_B),b_B),skvb))}/_{x_6}\})
\end{aligned}
\tag{3}
$$

Up to this point, each thread on the left hand side is simulated by the same process on the right hand side. The order of the transitions can change according to the partial order of the processes, but the resulting frames remain statically equivalent. The next step depends on the context. If the attacker returns correctly signed votes to the two honest voters and tells $V_C^{c_1,c_2}$ to go on (i.e. sends a message containing `true`), the processes can synchronize and go on. Otherwise at least one of them will block and they will be unable to synchronize.

If they are able to synchronize, they will output their unblinded vote to the collector ($V_C^{c_1,c_2}$ will send any message the attacker gives him). From this point on, the honest voters swap their roles, i.e. $V_B\ \{^a/_v\}$ simulates the behavior of $V_A\ \{^a/_v\}$. We obtain the following frames

$$
\begin{aligned}
\phi_l = \nu b_B.\nu b_A.\nu r_A.\nu r_B.(\{^{skvc}/_{x_1}\} | \{^{pka}/_{x_2}\} | \{^{y1}/_{x_3}\} | \{^{m2}/_{x_4}\} | \\
\{^{(pk(skva),sign(blind(commit(a,r_A),b_A),skva))}/_{x_5}\} | \\
\{^{(pk(skvb),sign(blind(commit(b,r_B),b_B),skvb))}/_{x_6}\} | \\
\{^{(commit(a,r_A),sign(commit(a,r_A),ska))}/_{x_7}\} | \\
\{^{(commit(b,r_B),sign(commit(b,r_B),ska))}/_{x_8}\} | \\
\{^{y3}/_{x_9}\}) | \{^{m3}/_{x_{10}}\})
\end{aligned}
\tag{4}
$$

```
(* private channels *)
ν privCh . ν pkaCh1 . ν pkaCh2 . ν skaCh .
ν skvaCh . ν skvbCh . ν skvcCh . ν chT .
(* administrators *)
(processK | processA | processA | processA |
processC | processC | processC |
processT | processT | processT |
(* voters *)
(let skvCh = skvaCh in let v = a in processV) |
(let skvCh = skvbCh in let v = b in processV) |
(let skvCh = skvcCh in let v = c in processV))
```

Process 7: The main process

$$\begin{aligned}\phi_r = \nu b_B.\nu b_A.\nu r_A.\nu r_B.(&\{^{skvc}/_{x_1}\} \mid \{^{pka}/_{x_2}\} \mid \{^{y1}/_{x_3}\} \mid \{^{m2}/_{x_4}\} \mid\\ &\{^{(pk(skva),sign(blind(commit(b,r_A),b_A),skva))}/_{x_5}\} \mid\\ &\{^{(pk(skvb),sign(blind(commit(a,r_B),b_B),skvb))}/_{x_6}\} \mid\\ &\{^{(commit(b,r_A),sign(commit(b,r_A),ska))}/_{x_7}\} \mid\\ &\{^{(commit(a,r_B),sign(commit(a,r_B),ska))}/_{x_8}\} \mid\\ &\{^{y3}/_{x_9}\}) \mid \{^{m3}/_{x_{10}}\})\end{aligned} \tag{5}$$

Then it depends again on the input of the attacker. If either of the honest voters gets a wrong input or the attacker decides to block $V_C^{c_1,c_2}$, the voters cannot synchronize. If both honest voters receive a correct input and if they can synchronize with $V_C^{c_1,c_2}$, they will reveal their random values with the corresponding l. This yields the following frames:

$$\begin{aligned}\phi'_l = \nu b_B.\nu b_A.\nu r_A.\nu r_B.\nu l_A.\nu l_B.(&\{^{skvc}/_{x_1}\} \mid \{^{pka}/_{x_2}\} \mid \{^{y1}/_{x_3}\} \mid \{^{m2}/_{x_4}\} \mid\\ &\{^{(pk(skva),sign(blind(commit(a,r_A),b_A),skva))}/_{x_5}\} \mid\\ &\{^{(pk(skvb),sign(blind(commit(b,r_B),b_B),skvb))}/_{x_6}\} \mid\\ &\{^{(commit(a,r_A),sign(commit(a,r_A),ska))}/_{x_7}\} \mid\\ &\{^{(commit(b,r_B),sign(commit(b,r_B),ska))}/_{x_8}\} \mid\\ &\{^{y3}/_{x_9}\} \mid \{^{m3}/_{x_{10}}\} \mid \{^{(l_A,r_A)}/_{x_{11}}\} \mid \{^{(l_B,r_B)}/_{x_{12}}\} \mid \{^{y6}/_{x_{13}}\})\end{aligned} \tag{6}$$

$$\begin{aligned}\phi'_r = \nu b_B.\nu b_A.\nu r_A.\nu r_B.\nu l_A.\nu l_B.(&\{^{skvc}/_{x_1}\} \mid \{^{pka}/_{x_2}\} \mid \{^{y1}/_{x_3}\} \mid \{^{m2}/_{x_4}\} \mid\\ &\{^{(pk(skva),sign(blind(commit(b,r_A),b_A),skva))}/_{x_5}\} \mid\\ &\{^{(pk(skvb),sign(blind(commit(a,r_B),b_B),skvb))}/_{x_6}\} \mid\\ &\{^{(commit(b,r_A),sign(commit(b,r_A),ska))}/_{x_7}\} \mid\\ &\{^{(commit(a,r_B),sign(commit(a,r_B),ska))}/_{x_8}\} \mid\\ &\{^{y3}/_{x_9}\} \mid \{^{m3}/_{x_{10}}\} \mid \{^{(l_A,r_A)}/_{x_{11}}\} \mid \{^{(l_B,r_B)}/_{x_{12}}\} \mid \{^{y6}/_{x_{13}}\})\end{aligned} \tag{7}$$

These frames are statically equivalent, which gives us the desired result. □

E Proof of Proposition 4 ("Okamoto $\in VI - PC$")

We us the following model for our proof:

The main process The main process (Process 7) shows how the participation processes (three voters, three administrators, three collectors, three talliers - one for each voter - and the keying process) are combined in parallel using private channels.

```
processK =
        (* private keys *)
        ν ska . ν skva . ν skvb . ν skvc.
        (* public keys *)
        let (pka, pkva, pkvb, pkvc)
        = (pk(ska), pk(skva), pk(skvb), pk(skvc)) in
        (* public key disclosure *)
        out(ch, pka).
        out(ch, pkva).out(ch, pkvb).out(ch, pkvc).
        (* register legitimate voters *)
        (out(privCh, pkva) | out(privCh, pkvb) |
         out(privCh, pkvc) | out(pkaCh1, pka) |
         out(pkaCh1, pka) | out(pkaCh1, pka) |
         out(pkaCh2, pka) | out(pkaCh2, pka) |
         out(pkaCh2, pka) | out(skaCh, ska) |
         out(skvaCh, skva) | out(skvbCh, skvb) |
         out(skvcCh, skvc))
```

Process 8: The key distribution process

```
processV =
        (* private key *)
        in(skvCh, skv).
        (* public key of administrator *)
        in(pkaCh1, pka).
        ν blinder . ν r . ν td .
        let commitedvote = tdcommit(v, r, td) in
        let blindedvote = blind(commitedvote, blinder) in
        out(ch1, (pk(skv), sign(blindedvote, skv))).
        in(ch2, m2).
        let result = checksign(m2, pka) in
        if result = blindedvote then
        let signedvote = unblind(m2, blinder) in
        synch 1.
        out(ch3, (commitedvote, signedvote)).
        out(chT, (v, r, commitedvote))
```

Process 9: The voting process

```
processVc1c2 =
        (* private key *)
        in(skvCh, skv).out(c1, skv).
        (* public key of administrator *)
        in(pkaCh1, pka).out(c1, pka).
        ν blinder .out(c1, blinder). ν r .out(c1, r).
        ν td .out(c1, td).
        let commitedvote = tdcommit(v, r, td) in
        let blindedvote = blind(commitedvote, blinder) in
        in(c2, y1).out(ch1, y1).
        in(ch2, m2).out(c1, m2).
        let result = checksign(m2, pka) in
        in(c2, y2).
        if y2 = true then
        let signedvote = unblind(m2, blinder) in
        synch 1.
        in(c2, y3).out(ch3, y3).
        in(c2, y4).out(chT, y4)
```

Process 10: The voting process under control of the attacker

```
processV =
        (* private key *)
        in(skvCh, skv).out(chc, skv).
        (* public key of administrator *)
        in(pkaCh1, pka).out(chc, pka).
        ν blinder . ν r . ν td .
        out(chc, blinder).out(chc, f(a, r, td, c)).
        out(chc, td).
        let commitedvote = tdcommit(a, r, td) in
        let blindedvote = blind(commitedvote, blinder) in
        out(ch1, (pk(skv), sign(blindedvote, skv))).
        out(chc, (pk(skv), sign(blindedvote, skv))).
        in(ch2, m2).
        let result = checksign(m2, pka) in
        if result = blindedvote then
        let signedvote = unblind(m2, blinder) in
        synch 1.
        out(ch3, (commitedvote, signedvote)).
        out(chc, (commitedvote, signedvote)).
        out(chT, (a, r, commitedvote)).
        out(chT, (c, f(a, r, td, c), commitedvote))
```

Process 11: The process V'

```
processC =
        (* administrator's public key *)
        in(pkaCh2, pka).
        synch 1.
        in(ch3, (m3, m4)).
        if checksign(m4, pka) = m3 then
        synch 2.
        out(ch, (m3, m4))
```

Process 12: The collector process

```
processT =
        synch 1.
        (* receiving the commitment *)
        in(chT, (vt, rt, xt)).
        synch 2.
        if open(xt, rt) = vt then
        out(ch, vt)
```

Process 13: The timeliness process

The keying process The keying process (Process 8) creates the private keys, distributes them over private channels and publishes the corresponding public keys. He also registers legitimate voters and sends this list to the administrators.

The voter The voter process (Process 9) receives the necessary keys and follows the nearly the same protocol as in the case of FOO, but he has to reveal the data necessary to open the commitment over a private channel to the timeliness member T.

A voter under control of the attacker A voter controlled by the attacker is modelled by Process 10. This process can be obtained by calculating $\texttt{processV}^{c_1,c_2}$ as defined in Definition 9.

The administrator The administrator is exactly the same as given in Section 4.4 (Process 3).

The collector The collector (Process 12) receives the administrator's public key, which he then uses to verify the signature on incoming commitments. If the signature is correct, he publishes the commitment on a public channel.

The timeliness member The timeliness process (Process 13) receives the vote, the corresponding commitment and the randomness over a private channel. He verifies the correctness of the data and then publishes the vote.

Proof. We will show that there exists a closed plain process V' such that

$$V'^{\setminus out(chc,\cdot)} \approx_l V_A\{^a/_v\}$$

and

$$\begin{gathered}\nu pkaCh1.\nu chT.(V_A\{^c/_v\}^{chc}|V_B\{^a/_v\}|V_C^{c_1,c_2}| \\ \texttt{processK}|\texttt{processT}|\texttt{processT}|\texttt{processT}) \\ \approx_l \\ \nu pkaCh1.\nu chT.(V'|V_B\{^c/_v\}|V_C^{c_1,c_2}| \\ \texttt{processK}|\texttt{processT}|\texttt{processT}|\texttt{processT})\end{gathered}$$

where $V_A = \texttt{processV}\{^{skvaCh}/_{skvCh}\}$, $V_B = \texttt{processV}\{^{skvbCh}/_{skvCh}\}$ and $V_C^{c_1,c_2} = \texttt{processVc1c2}\{^{skvcCh}/_{skvCh}\}$. As in the case of Fujioka et al., our proof will technically only be correct for three voters, but it can be repeated for more voters.

The first equivalence is easy to see by removing all $\texttt{out}(\texttt{chc}, \_)$ from V'. The resulting process is apparently equal to the original voting process V_A.

The second equivalence is more difficult to prove. The beginning is the same as in the proof of vote-independence for FOO. We will denote the left hand side process P and the right hand side process Q. Both sides start by distributing the keys. The public key of the administrator is passed through a private channel which yields an internal reduction. Again, for better readability, we will not detail this part. $V_C^{c_1,c_2}$ will output the keys he receives and forward the attacker's input to the administrator and resend the answer to the administrator (which is also part of the attacker). The honest voters construct the blinded and committed vote and send them to the administrator. During this process, V_A outputs his secret values and inputs, where V' fakes these values.

$$
\begin{aligned}
P \quad & \xrightarrow{\texttt{in}(skvaCh,skva)} \xrightarrow{\nu x_1.\texttt{out}(chc,x_1)} \nu x_1.(P_1 \mid \{^{skva}/_{x_1}\}) \\
& \xrightarrow{\texttt{in}(skvbCh,skvb)} \xrightarrow{\texttt{in}(skvcCh,skvc)} \rightarrow^* \xrightarrow{\nu x_2.\texttt{out}(c1,x_2)} \nu\tilde{x}.(P_2 \mid \{^{skva}/_{x_1}\} \mid \{^{skvc}/_{x_2}\}) \\
& \xrightarrow{\nu x_3.\texttt{out}(c1,x_3)} \xrightarrow{\nu x_4.\texttt{out}(chc,x_4)} \\
& \qquad \nu\tilde{x}.(P_3 \mid \{^{skva}/_{x_1}\} \mid \{^{skvc}/_{x_2}\} \mid \{^{pk(ska)}/_{x_3}\} \mid \{^{pk(ska)}/_{x_4}\}) \\
& \xrightarrow{\nu x_5.\texttt{out}(chc,x_5)} \xrightarrow{\nu x_6.\texttt{out}(chc,x_6)} \xrightarrow{\nu x_7.\texttt{out}(chc,x_7)} \nu\tilde{x}.(P_4 \mid \\
& \quad \{^{skva}/_{x_1}\} \mid \{^{skvc}/_{x_2}\} \mid \{^{pk(ska)}/_{x_3}\} \mid \{^{pk(ska)}/_{x_4}\} \mid \{^{b_A}/_{x_5}\} \mid \{^{r_A}/_{x_6}\} \mid \{^{td_A}/_{x_7}\}) \\
& \xrightarrow{\nu x_8.\texttt{out}(ch1,x_8)} \xrightarrow{\nu x_9.\texttt{out}(chc,x_9)} \nu\tilde{x}.(P_5 \mid \\
& \quad \{^{skva}/_{x_1}\} \mid \{^{skvc}/_{x_2}\} \mid \{^{pk(ska)}/_{x_3}\} \mid \{^{pk(ska)}/_{x_4}\} \mid \{^{b_A}/_{x_5}\} \mid \{^{r_A}/_{x_6}\} \mid \{^{td_A}/_{x_7}\} \mid \\
& \qquad \{^{(pk(skva),sign(blind(tdcommit(c,r_A,td_A),b_A),skva))}/_{x_8}\} \mid \{^{x_8}/_{x_9}\}) \\
& \xrightarrow{\nu x_{10}.\texttt{out}(ch1,x_{10})} \nu\tilde{x}.(P_6 \mid \\
& \quad \{^{skva}/_{x_1}\} \mid \{^{skvc}/_{x_2}\} \mid \{^{pk(ska)}/_{x_3}\} \mid \{^{pk(ska)}/_{x_4}\} \mid \{^{b_A}/_{x_5}\} \mid \{^{r_A}/_{x_6}\} \mid \{^{td_A}/_{x_7}\} \mid \\
& \qquad \{^{(pk(skva),sign(blind(tdcommit(c,r_A,td_A),b_A),skva))}/_{x_8}\} \mid \{^{x_8}/_{x_9}\} \mid \\
& \qquad \{^{(pk(skva),sign(blind(tdcommit(a,r_B,td_B),b_B),skvb))}/_{x_{10}}\}) \\
& \xrightarrow{\texttt{in}(c2,y1)} \xrightarrow{\nu x_{11}.\texttt{out}(ch1,x_{11})} \nu\tilde{x}.(P_7 \mid \\
& \quad \{^{skva}/_{x_1}\} \mid \{^{skvc}/_{x_2}\} \mid \{^{pk(ska)}/_{x_3}\} \mid \{^{pk(ska)}/_{x_4}\} \mid \{^{b_A}/_{x_5}\} \mid \{^{r_A}/_{x_6}\} \mid \{^{td_A}/_{x_7}\} \mid \\
& \qquad \{^{(pk(skva),sign(blind(tdcommit(c,r_A,td_A),b_A),skva))}/_{x_8}\} \mid \{^{x_8}/_{x_9}\} \mid \\
& \qquad \{^{(pk(skva),sign(blind(tdcommit(a,r_B,td_B),b_B),skvb))}/_{x_{10}}\} \mid \{^{y1}/_{x_{11}}\})
\end{aligned}
$$

Similarly

$$
\begin{aligned}
Q \xrightarrow{\mathtt{in}(skvaCh,skva)} & \xrightarrow{\nu x_1.\mathtt{out}(chc,x_1)} \nu x_1.(Q_1 | \{^{skva}/_{x_1}\}) \\
& \xrightarrow{\mathtt{in}(skvbCh,skvb)} \xrightarrow{\mathtt{in}(skvcCh,skvc)} \rightarrow^* \xrightarrow{\nu x_2.\mathtt{out}(c1,x_2)} \nu\tilde{x}.(Q_2 | \{^{skva}/_{x_1}\} | \{^{skvc}/_{x_2}\}) \\
& \xrightarrow{\nu x_3.\mathtt{out}(c1,x_3)} \xrightarrow{\nu x_4.\mathtt{out}(chc,x_4)} \\
& \qquad \nu\tilde{x}.(Q_3 | \{^{skva}/_{x_1}\} | \{^{skvc}/_{x_2}\} | \{^{pk(ska)}/_{x_3}\} | \{^{pk(ska)}/_{x_4}\}) \\
& \xrightarrow{\nu x_5.\mathtt{out}(chc,x_5)} \xrightarrow{\nu x_6.\mathtt{out}(chc,x_6)} \xrightarrow{\nu x_7.\mathtt{out}(chc,x_7)} \nu\tilde{x}.(Q_4 | \{^{skva}/_{x_1}\} | \\
& \qquad \{^{skvc}/_{x_2}\} | \{^{pk(ska)}/_{x_3}\} | \{^{pk(ska)}/_{x_4}\} | \{^{b_A}/_{x_5}\} | \{^{f(a,r_A,td_A,c)}/_{x_6}\} | \{^{td_A}/_{x_7}\}) \\
& \xrightarrow{\nu x_8.\mathtt{out}(ch1,x_8)} \xrightarrow{\nu x_9.\mathtt{out}(chc,x_9)} \nu\tilde{x}.(Q_5 | \{^{skva}/_{x_1}\} | \\
& \qquad \{^{skvc}/_{x_2}\} | \{^{pk(ska)}/_{x_3}\} | \{^{pk(ska)}/_{x_4}\} | \{^{b_A}/_{x_5}\} | \{^{f(a,r_A,td_A,c)}/_{x_6}\} | \{^{td_A}/_{x_7}\} | \\
& \qquad\qquad \{^{(pk(skva),sign(blind(tdcommit(a,r_A,td_A),b_A),skva))}/_{x_8}\} | \{^{x_8}/_{x_9}\}) \\
& \xrightarrow{\nu x_{10}.\mathtt{out}(ch1,x_{10})} \nu\tilde{x}.(Q_6 | \{^{skva}/_{x_1}\} | \\
& \qquad \{^{skvc}/_{x_2}\} | \{^{pk(ska)}/_{x_3}\} | \{^{pk(ska)}/_{x_4}\} | \{^{b_A}/_{x_5}\} | \{^{f(a,r_A,td_A,c)}/_{x_6}\} | \{^{td_A}/_{x_7}\} | \\
& \qquad\qquad \{^{(pk(skva),sign(blind(tdcommit(a,r_A,td_A),b_A),skva))}/_{x_8}\} | \{^{x_8}/_{x_9}\} | \\
& \qquad\qquad\qquad \{^{(pk(skva),sign(blind(tdcommit(c,r_B,td_B),b_B),skvb))}/_{x_{10}}\}) \\
& \xrightarrow{\mathtt{in}(c2,y1)} \xrightarrow{\nu x_{11}.\mathtt{out}(ch1,x_{11})} \nu\tilde{x}.(Q_7 | \{^{skva}/_{x_1}\} | \\
& \qquad\qquad \{^{skvc}/_{x_2}\} | \{^{pk(ska)}/_{x_3}\} | \{^{pk(ska)}/_{x_4}\} | \{^{b_A}/_{x_5}\} | \{^{r_A}/_{x_6}\} | \{^{td_A}/_{x_7}\} | \\
& \qquad\qquad\qquad \{^{(pk(skva),sign(blind(tdcommit(a,r_A,td_A),b_A),skva))}/_{x_8}\} | \{^{x_8}/_{x_9}\} | \\
& \qquad\qquad\qquad \{^{(pk(skva),sign(blind(tdcommit(c,r_B,td_B),b_B),skvb))}/_{x_{10}}\} | \{^{y1}/_{x_{11}}\})
\end{aligned}
$$

We can argue that at this point the obtained frames are statically equivalent. In particular the attacker obtains in both cases

$$\mathtt{open}(\mathtt{unblind}(\mathtt{checksign}(\mathtt{proj}_2(x_8), \mathtt{pk}(x_1)), x_5), x_6) = c$$

when he tries to open the commitment due to the trap-door and the faked randomness.

The next step depends on the context. If the attacker returns correctly signed votes to the two honest voters and tells $V_C^{c_1,c_2}$ to go on (i.e. sends a message containing `true`), the processes can synchronize and go on. Otherwise at least one of them will block and they will be unable to synchronize.

If they are able to synchronize, they will output their votes to the collector ($V_C^{c_1,c_2}$ will send any message the attacker gives him) and send the secret values to the timeliness member over a private channel. We obtain the following frames:

$$
\begin{aligned}
\phi_l = \nu\tilde{x}.(&\{^{skva}/_{x_1}\} | \{^{skvc}/_{x_2}\} | \{^{pk(ska)}/_{x_3}\} | \{^{pk(ska)}/_{x_4}\} | \{^{b_A}/_{x_5}\} | \{^{r_A}/_{x_6}\} | \\
& \{^{td_A}/_{x_7}\} | \{^{(pk(skva),sign(blind(tdcommit(c,r_A,td_A),b_A),skva))}/_{x_8}\} | \{^{x_8}/_{x_9}\} | \\
& \{^{(pk(skva),sign(blind(tdcommit(a,r_B,td_B),b_B),skvb))}/_{x_{10}}\} | \{^{y1}/_{x_{11}}\} | \\
& \{^{(tdcommit(c,r_A,td_A),sign((tdcommit(c,r_A,td_A),ska)}/_{x_{12}}\} | \\
& \{^{(tdcommit(a,r_B,td_B),sign((tdcommit(a,r_B,td_B),ska)}/_{x_{13}}\} | \\
& \{^{(c,r_A,tdcommit(c,r_A,td_A))}/_{x_{14}}\} | \\
& \{^{a}/_{x_{15}}\} | \{^{c}/_{x_{16}}\})
\end{aligned}
\tag{8}
$$

$$
\begin{aligned}
\phi_r = \nu\tilde{x}.(&\{^{skva}/_{x_1}\} | \{^{skvc}/_{x_2}\} | \{^{pk(ska)}/_{x_3}\} | \{^{pk(ska)}/_{x_4}\} | \{^{b_A}/_{x_5}\} | \\
& \{^{f(a,r_A,td_A,c)}/_{x_6}\} | \{^{td_A}/_{x_7}\} | \\
& \{^{(pk(skva),sign(blind(tdcommit(a,r_A,td_A),b_A),skva))}/_{x_8}\} | \{^{x_8}/_{x_9}\} | \\
& \{^{(pk(skva),sign(blind(tdcommit(c,r_B,td_B),b_B),skvb))}/_{x_{10}}\} | \{^{y1}/_{x_{11}}\} | \\
& \{^{(tdcommit(a,r_A,td_A),sign((tdcommit(a,r_A,td_A),ska)}/_{x_{12}}\} | \\
& \{^{(tdcommit(c,r_B,td_B),sign((tdcommit(c,r_B,td_B),ska)}/_{x_{13}}\} | \\
& \{^{(c,f(a,r_A,td_A,c),tdcommit(a,r_A,td_A))}/_{x_{14}}\} | \\
& \{^{a}/_{x_{15}}\} | \{^{c}/_{x_{16}}\})
\end{aligned}
\tag{9}
$$

```
(* private channels *)
ν privChM1 . ν privChM2 . ν privChM3.
ν privChRM1 . ν privChRM2 . ν privChRM3 .
ν privChR1 . ν privChR2 . ν privChR3.
(* voting machine *)
(processM |
(* RNGs *)
(let privChM = privChRM1 in
  let privChV = privChR1 in processRNG) |
(let privChM = privChRM2 in
  let privChV = privChR2 in processRNG) |
(let privChM = privChRM3 in
  let privChV = privChR3 in processRNG) |
(* voters *)
(let privChM = privChM1 in
  let privChRNG = privChR1 in
   let v = p1 in processV) |
(let privChM = privChM2 in
  let privChRNG = privChR2 in
   let v = p2 in processV) |
(let privChM = privChM3 in
  let privChRNG = privChR3 in
   let v = p3 in processV))
```

Process 14: The main process

These frames are statically equivalent. If the attacker sent a correct input to the third timeliness process, T will put out the corresponding vote. Note that if the attacker copies V_A's vote using the possibly faked credentials, T will always output c.

More generally, the attacker cannot use T to obtain frames that are not statically equivalent as this would imply that this difference could also be made on the frames before by opening a commitment - something the attacker could do by himself. Intuitively having access to the "Timeliness" oracle does not help the attacker, as the oracle performs only operations the attacker could have done by himself. □

F Proof of Proposition 5 ("Bingo Voting $\in VI - AC$")

To model the voting booth, we use private channels between the voting machine and the voter, the voter and the RNG, and between the RNG and the voting machine. To achieve better readability we use the macros `for` and `parfor` (similar to [17]) where e.g. `for (i = 1 to 2) out(ch, i)` corresponds to `out(ch, 1).out(ch, 2)` and `parfor (i = 1 to 2) out(ch, i)` corresponds to `(out(ch, 1) | out(ch, 2))`. Additionally we use a function `choose` where

$$\texttt{choose}(p_i, p_j, x, y) = \begin{cases} x \text{ if } i = j \\ y \text{ else} \end{cases}$$

Our model depends on two parameters: k (the number of voters) and l (the number of candidates).

The main process The main process (Process 14) sets up the private channels and executes the participation processes (the voting machine, three voters, and three RNGs) in parallel.

The RNG process The RNG (Process 15) generates a random number and sends it to the voting machine and the voter over private channels.

```
processRNG =
        (* generate random number *)
        ν r.
        (* output to voting machine *)
        out(privChM, r).
        (* output to voter *)
        out(privChV, r)
```

Process 15: The random number generator (RNG)

```
processV =
        (* voting *)
        out(privChM, v).
        (* receipt *)
        for (i = 1 to l)
         in(privChM, receipt_i)
        (* random value, to verify receipt *)
        in(privChRNG, r)
```

Process 16: The voting process

```
processVc1c2 =
        (* voting *)
        in(c1, x1).
        out(privChM, x1).
        (* receipt *)
        for (i = 1 to l)
         in(privChM, receipt_i).
         out(c2, receipt_i)
        (* random value, to verify receipt *)
        in(privChRNG, r).
        out(c2, r)
```

Process 17: The process $V^{c1,c2}$

```
processV =
        (* voting *)
        in(c1, x1).
        out(privChM, v).
        (* random value, to verify receipt *)
        in(privChRNG, r).
        (* receipt *)
        for (i = 1 to l)
         in(privChM, receipt_i).
         out(c2, receipt_i)
        (* output correct random value *)
        for (i = 1 to l)
         if(x1 = p_i) then out(c2, receipt_i)
```

Process 18: The process V'

```
processM' = in(privChV, v). in(privChR, r).
        for (j = 1 to l)
         let receipt_j = choose_l(v, r, n_{i,j}) in
          out(privChV, receipt_j);
        synch1.
        (* output result *)
        out(ch, v)
        (* output receipts *)
        for (j = 1 to l)
         out(ch, receipt_j)
        (* output unused dummy votes *)
        parfor (j = 1 to l)
         if v_i ≠ p_j then out(ch, ((n_{i,j}, p_j),
          commit((n_{i,j}, p_j), r_{i,j}), r_{i,j}))

processM =
        (* prepare dummy votes *)
        for (i = 1 to k)
         for (j = 1 to l)
          νn_{i,j}.νr_{i,j}.
        parfor (i = 1 to k)
         parfor (j = 1 to l)
          out(ch, commit((n_{i,j}, p_j), r_{i,j}))
         (* voting *)
         let privChV = privChM_i in
          let privChR = privChRM_i in processM'
```

Process 19: The voting machine

The voter process The voter (Process 16) sends his vote to the voting machine, receives the random number from the RNG and the receipt.

The $V^{c1,c2}$ process The $V^{c1,c2}$ process (Process 17) simulates a coerced voter. He follows the same protocol, but votes for the candidate the coercer tells him and forwards the receipt and the random number.

The voting machine process The voting machine (Process 19) generates the dummy votes and publishes the corresponding commitments. Then k subprocesses interact with the voters, i.e. create the receipts. After all voting has been done, they publish the result, the receipts and the unused dummies in random order.

Proof. To show that Bingo Voting ensures VI-AC, we will show that for any $C = \nu c_1.\nu c_2.(\_|P)$ satisfying $\tilde{n} \cap fn(C) = \emptyset$ and

$$\begin{array}{c} \nu\tilde{\mathtt{ch}}.(C\,[V_A\,\{^{?}/_{v}\}^{c_1,c_2}]\,|V_B\,\{^{a}/_{v}\}\,|V_C^{c_1,c_2}|M_{k=3,l=2}|R_A|R_B|R_C) \\ \approx_l \\ \nu\tilde{\mathtt{ch}}.(V_A\,\{^{c}/_{v}\}^{chc}\,|V_B\,\{^{a}/_{v}\}\,|V_C^{c_1,c_2}|M_{k=3,l=2}|R_A|R_B|R_C) \end{array}$$

we have

$$C\,[V']^{\setminus out(chc,\cdot)} \approx_l V_A\,\{^{a}/_{v}\}$$

and

$$\begin{array}{c} \nu\tilde{\mathtt{ch}}.(C\,[V_A\,\{^{?}/_{v}\}^{c_1,c_2}]\,|V_B\,\{^{a}/_{v}\}\,|V_C^{c_3,c_4}|M_{k=3,l=2}|R_A|R_B|R_C) \\ \approx_l \\ \nu\tilde{\mathtt{ch}}.(C\,[V']\,|V_B\,\{^{c}/_{v}\}\,|V_C^{c_3,c_4}|M_{k=3,l=2}|R_A|R_B|R_C) \end{array} \tag{10}$$

where

$$\begin{array}{rcl} \nu\tilde{\mathtt{ch}} & = & \nu\mathtt{privChM}_1.\nu\mathtt{privChM}_2.\nu\mathtt{privChM}_3. \\ & & \nu\mathtt{privChRM}_1.\nu\mathtt{privChRM}_2.\nu\mathtt{privChRM}_3. \\ & & \nu\mathtt{privChR}_1.\nu\mathtt{privChR}_2.\nu\mathtt{privChR}_3 \\ V_A & = & \mathtt{processV}\,\{^{privChM_1}/_{privChM}, {}^{privChR_1}/_{privChRNG}\} \\ V_B & = & \mathtt{processV}\,\{^{privChM_2}/_{privChM}, {}^{privChR_2}/_{privChRNG}\} \\ V_C & = & \mathtt{processV}\,\{^{privChM_3}/_{privChM}, {}^{privChR_3}/_{privChRNG}\} \\ R_A & = & \mathtt{processD}\,\{^{privChRM_1}/_{privChM}, {}^{privChR_1}/_{privChV}\} \\ R_B & = & \mathtt{processD}\,\{^{privChRM_2}/_{privChM}, {}^{privChR_2}/_{privChV}\} \\ R_C & = & \mathtt{processD}\,\{^{privChRM_3}/_{privChM}, {}^{privChR_3}/_{privChV}\} \end{array}$$

This is technically only a valid proof for two honest voters and two candidates, however a similar proof can be done for an arbitrary number of voters and candidates.

It is easy to see that $C\,[V']^{\setminus out(chc,\cdot)} \approx_l V_A\,\{^{a}/_{v}\}$ holds. If we ignore out all inputs of V' on $\mathtt{c1}$ and all outputs on $\mathtt{c2}$, we obtain V_A.

We now have to show that the last equivalence holds. As before, we will denote the left hand side P and the right hand side Q. For better readability we concentrate on the important steps. At the beginning the voting machine publishes all commitments; then the voters enter the voting booth and vote. We consider only two candidates, thus $c = p_1$ or $c = p_2$ (similarly for a). Here we will look at the case $c = p_2$ and $a = p_1$, the other cases are similar. Since all communication inside the booth takes place over internal channels which yield internal reductions (we do not detail this part) and first condition on C ensures that the targeted voter is forced to vote c, we obtain the following two frames:

$$\begin{array}{rl} \phi_l = \nu\tilde{x}.\tilde{r}.\tilde{n}.(\{^{commit((n_{1,1},p_1),r_{1,1})}/_{x_1}\} \mid \{^{commit((n_{2,1},p_1),r_{2,1})}/_{x_2}\} \mid & \\ \{^{commit((n_{3,1},p_1),r_{3,1})}/_{x_3}\} \mid \{^{commit((n_{1,2},p_2),r_{1,2})}/_{x_4}\} \mid & \\ \{^{commit((n_{2,2},p_2),r_{2,2})}/_{x_5}\} \mid \{^{commit((n_{3,2},p_2),r_{3,2})}/_{x_6}\} \mid & \\ \{^{n_{1,1}}/_{x_7}\} \mid \{^{r_1}/_{x_8}\} \mid \{^{r_1}/_{x_9}\}) & \end{array} \tag{11}$$

$$\begin{aligned}\phi_r = \nu\tilde{x}.\tilde{r}.\tilde{n}.(&\{commit((n_{1,1},p_1),r_{1,1})/x_1\} \mid \{commit((n_{2,1},p_1),r_{2,1})/x_2\} \mid \\ &\{commit((n_{3,1},p_1),r_{3,1})/x_3\} \mid \{commit((n_{1,2},p_2),r_{1,2})/x_4\} \mid \\ &\{commit((n_{2,2},p_2),r_{2,2})/x_5\} \mid \{commit((n_{3,2},p_2),r_{3,2})/x_6\} \mid \\ &\{r_1/x_7\} \mid \{n_{1,1}/x_8\} \mid \{n_{1,1}/x_9\})\end{aligned} \tag{12}$$

Obviously both frames are statically equivalent. Note that at this point no information about V_B and his vote is available. Now the attacker has to vote himself, otherwise the voting machines will be unable to synchronize and block. However he cannot relate his vote in any way to V_A's vote, as the receipts are meaningless to him and he has to submit his vote in clear. Suppose that he votes for p_1 (the other case is similar). We obtain the folowing final frames.

$$\begin{aligned}\phi'_l = \nu\tilde{x}.\tilde{r}.\tilde{n}.(&\{commit((n_{1,1},p_1),r_{1,1})/x_1\} \mid \{commit((n_{2,1},p_1),r_{2,1})/x_2\} \mid \\ &\{commit((n_{3,1},p_1),r_{3,1})/x_3\} \mid \{commit((n_{1,2},p_2),r_{1,2})/x_4\} \mid \\ &\{commit((n_{2,2},p_2),r_{2,2})/x_5\} \mid \{commit((n_{3,2},p_2),r_{3,2})/x_6\} \mid \\ &\{n_{1,1}/x_7\} \mid \{r_1/x_8\} \mid \{r_1/x_9\} \mid \\ &\{r_3/x_{10}\} \mid \{n_{3,2}/x_{11}\} \mid \{r_3/x_{12}\} \mid \\ &\{p_1/x_{13}\} \mid \{p_1/x_{14}\} \mid \{p_2/x_{15}\} \mid \\ &\{n_{1,1}/x_{16}\} \mid \{r_1/x_{17}\} \mid \{r_2/x_{18}\} \mid \{n_{2,2}/x_{19}\} \mid \{r_3/x_{20}\} \mid \{n_{3,2}/x_{21}\} \mid \\ &\{((n_{2,1},p_1),commit((n_{2,1},p_1),r_{2,1}),r_{2,1})/x_{22}\} \mid \\ &\{((n_{3,1},p_1),commit((n_{3,1},p_1),r_{3,1}),r_{3,1})/x_{23}\} \mid \\ &\{((n_{1,2},p_2),commit((n_{1,2},p_2),r_{1,2}),r_{1,2})/x_{24}\})\end{aligned} \tag{13}$$

$$\begin{aligned}\phi'_r = \nu\tilde{x}.\tilde{r}.\tilde{n}.(&\{commit((n_{1,1},p_1),r_{1,1})/x_1\} \mid \{commit((n_{2,1},p_1),r_{2,1})/x_2\} \mid \\ &\{commit((n_{3,1},p_1),r_{3,1})/x_3\} \mid \{commit((n_{1,2},p_2),r_{1,2})/x_4\} \mid \\ &\{commit((n_{2,2},p_2),r_{2,2})/x_5\} \mid \{commit((n_{3,2},p_2),r_{3,2})/x_6\} \mid \\ &\{r_1/x_7\} \mid \{n_{1,2}/x_8\} \mid \{n_{1,2}/x_9\} \mid \\ &\{r_3/x_{10}\} \mid \{n_{3,2}/x_{11}\} \mid \{r_3/x_{12}\} \mid \\ &\{p_1/x_{13}\} \mid \{p_2/x_{14}\} \mid \{p_1/x_{15}\} \mid \\ &\{r_1/x_{16}\} \mid \{n_{1,2}/x_{17}\} \mid \{n_{2,1}/x_{18}\} \mid \{r_2/x_{19}\} \mid \{r_3/x_{20}\} \mid \{n_{3,2}/x_{21}\} \mid \\ &\{((n_{1,1},p_1),commit((n_{1,1},p_1),r_{1,1}),r_{1,1})/x_{22}\} \mid \\ &\{((n_{3,1},p_1),commit((n_{3,1},p_1),r_{3,1}),r_{3,1})/x_{23})\} \mid \\ &\{((n_{2,2},p_2),commit((n_{2,2},p_2),r_{2,2}),r_{2,2})/x_{24})\})\end{aligned} \tag{14}$$

The election outcome and the data published by the voting machine do not help the attacker in distinguishing both cases. He can verify if the receipt by the coerced voter was correct (which it is), but he still does not know if the numbers on the receipt are unrevealed commitments or fresh random numbers. □

G Proof of Proposition 6 ("Lee et al. $\notin VI$")

We use the following model in applied pi:

The main process The main process (Process 20) sets up the private channels and executes the participating processes (three voters, three mixers, three talliers - one for each voter - and the keying process).

The keying process The keying process (Process 21) creates the private keys, distributes them over private channels and publishes the corresponding public keys. Each voter is equipped with his private key, the tallier's public key and the public key of his trusted device.

The voter The voter process (Process 22) receives the necessary keys. Then he encrypts his votes, signs it and sends it over a private channel to his trusted device. When he receives the answer, he checks the DVP, double signs the ballot and sends it to the mixer (which corresponds to the publication on the bulletin board).

```
(* private channels *)
ν sktCh . ν pktCh . ν votCh .
ν skvaCh . ν skvbCh . ν skvcCh .
ν skdaCh . ν skdbCh . ν skdcCh .
ν pkvaCh . ν pkvbCh . ν pkvcCh .
ν pkdaCh . ν pkdbCh . ν pkdcCh .
(* administrators *)
(processK | processM | processM | processM |
processT | processT | processT |
(* voters *)
(let skvCh = skvaCh in let pkdCh = pkdaCh in
 let v = a in processV) |
(let skvCh = skvbCh in let pkdCh = pkdbCh in
 let v = b in processV) |
(let skvCh = skvcCh in let pkdCh = pkdcCh in
 let v = c in processV) |
(* trusted devices *)
(let skdCh = skdaCh in
 let pkvCh = pkvaCh in processD) |
(let skdCh = skdbCh in
 let pkvCh = pkvbCh in processD) |
(let skdCh = skdcCh in
 let pkvCh = pkvcCh in processD))
```

Process 20: The main process

The trusted device The trusted device (Process 23) receives his private and the voter's public key. When he receives an encrypted ballot from the voter, he checks the signature, re-encrypts the message, signs it and establishes a DVP to prove the correctness of his re-encryption. The result is send back to the voter.

A voter under control of the attacker Process 24 models a voter controlled by the attacker. This process is obtained when calculating $\texttt{processV}^{c_1,c_2}$ as defined in Definition 9. Note that his trusted device is not assumed to be corrupted.

The mixer The mixer (Process 26) receives the key pair corresponding to a legitimate voter (the public keys of the voter and his trusted device). When receives ballot, he checks the signatures and mixes the now unsigned ballot with the other ballots. This is modeled using a synchronization point, a re-encryption and an anonymous channel.

The tallier The collector (Process 27) receives his private key, which he then uses to decrypt the incoming mixed ballots. He publishes the result on a public channel (which emulates the bulletin board).

Proof. In our model this can be seen as follows. We will suppose that all authorities are honest and that all key distribution channels are private. We will show that

$$\begin{array}{c} \nu\tilde{\texttt{ch}}.(V_A\{^a/_v\}|V_B\{^b/_v\}|V_C^{c_1,c_2}|D_A|D_B|D_C|A) \\ \not\approx_l \\ \nu\tilde{\texttt{ch}}.(V_A\{^b/_v\}|V_B\{^a/_v\}|V_C^{c_1,c_2}|D_A|D_B|D_C|A) \end{array} \tag{15}$$

```
processK =
        (* private keys *)
        ν skt.
        ν skva.ν skvb.ν skvc.
        ν skda.ν skdb.ν skdc.
        (* public keys *)
        let (pkt, pkva, pkvb, pkvc, pkda, pkdb, pkdc)
        = (pk(skt), pk(skva), pk(skvb), pk(skvc),
           pk(skda), pk(skdb), pk(skdc)) in
        (* public key disclosure *)
        out(ch, pkt).
        out(ch, pkva).out(ch, pkvb).out(ch, pkvc).
        out(ch, pkda).out(ch, pkdb).out(ch, pkdc).
        (* distribute keys: *)
        (* voters *)
        (out(skvaCh, skva) | out(skvbCh, skvb) |
         out(skvcCh, skvc) | out(pkdaCh, pkda) |
         out(pkdbCh, pkdb) | out(pkdcCh, pkdc) |
         out(pktCh, pkt) | out(pktCh, pkt) |
         out(pktCh, pkt) |
        (* trusted devices *)
         out(skdaCh, skva) | out(skdbCh, skvb) |
         out(skdcCh, skvc) | out(pkvaCh, pkva) |
         out(pkvbCh, pkvb) | out(pkvcCh, pkvc) |
        (* mixers *)
         out(votCh, (pkva, pkda)) |
         out(votCh, (pkvb, pkdb)) |
         out(votCh, (pkvc, pkdc)) |
        (* talliers*)
         out(sktCh, skt) | out(sktCh, skt) |
         out(sktCh, skt))
```

Process 21: The key distribution process

```
processV =
        (* private key *)
        in(skvCh, skv).
        (* public keys of the trusted device
           and the tallier *)
        in(pkdCh, pubkd).in(pktCh, pubkt).
        synch 1. ν r.
        let e = penc(v, pubkt, r) in
        out(chD, (pk(skv), e, sign(e, skv))).
        in(chD, m2).
        let (re, sd, dvpV) = m2 in
        if checkdvp(dvpV, e, re, pk(skv)) = ok then
        if checksign(sd, pubkd) = re then
        out(ch1, (re, sign(sd, skv)))
```

Process 22: The voting process

```
processD =
        (* private key *)
        in(skdCh, skd).
        (* public key of the voter *)
        in(pkvCh, pubkv).
        synch 1.
        in(chD, m1).
        let (pubv, enc, sig) = m1 in
        if pubv = pubkv then
        if checksign(sig, pubkv) = enc then
        ν r1.
        let reenc = rencrypt(enc, r1) in
        let signD = sign(reenc, skd) in
        let dvpD = dvp(enc, reenc, r1, pubkv) in
        out(chD, (reenc, signD, dvpD))
```

Process 23: The trusted device process

```
processVc1c2 =
        (* private key *)
        in(skvCh, skv).out(c1, skv).
        (* public keys of the trusted device
           and the tallier *)
        in(pkdCh, pubkd).out(c1, pubkd).
        in(pktCh, pubkt).out(c1, pubkt).
        synch 1. ν r.out(c1, r).
        let e = penc(v, pubkt, r) in
        in(c2, m1).out(chD, m1).
        in(chD, m2).out(c1, m2).
        let (re, sd, dvpV) = m2 in
        in(c2, m3).if m3 = true then
        in(c2, m4).if m4 = true then
        in(c2, m5).out(ch1, m5)
```

Process 24: The voting process under control of the attacker

```
processV' =
        (* private key *)
        in(skvCh, skv).out(c1, skv).
        (* public keys of the trusted device
           and the tallier *)
        in(pkdCh, pubkd).out(c1, pubkd).
        in(pktCh, pubkt).out(c1, pubkt).
        synch 1. ν r.out(c1, r).
        let e = penc(v, pubkt, r) in
        in(c2, m1).out(chD, (pk(skv), e, sign(e, skv))).
        let (pka, ea, sa) = m1 in
        in(chD, m2).
        let (re, sd, dvpV) = m2 in
        if checkdvp(dvpV, e, re, pk(skv)) = ok then
        if checksign(sd, pubkd) then
        let fk = dvp(ea, re, r', skv) in
        ν r'. out(c1, (re, sd, fk)).
        in(c2, m3).if m3 = true then
        in(c2, m4).if m4 = true then
        in(c2, m5).out(ch1, (re, sign(sd, skv)))
```

Process 25: The voting process resisting coercion

```
processM =
        (* register legitimate voters *)
        in(votCh, (pubkv, pubkd)).
        synch 1.
        in(ch1, m1).
        let (enc, sig) = m1 in
        if checksign(checksign(sig, pubkv), pubkd) = enc
         then synch 2. ν r2.
        out(ch2, rencrypt(enc, r2))
```

Process 26: The mixer process

```
processT =
        (* tallier's secret key *)
        in(sktCh, skt).
        synch 1.
        in(ch2, m).
        out(ch, decrypt(m, skt))
```

Process 27: The tallier process

where

$$\begin{aligned}
\nu\tilde{\mathtt{ch}} &= \nu\mathtt{sktCh}.\nu\mathtt{pktCh}.\nu\mathtt{votCh}.\\
&\quad \nu\mathtt{skvaCh}.\nu\mathtt{skvbCh}.\nu\mathtt{skvcCh}.\nu\mathtt{skdaCh}.\nu\mathtt{skdbCh}.\nu\mathtt{skdcCh}.\\
&\quad \nu\mathtt{pkvaCh}.\nu\mathtt{pkvbCh}.\nu\mathtt{pkvcCh}.\nu\mathtt{pkdaCh}.\nu\mathtt{pkdbCh}.\nu\mathtt{pkdcCh}\\
A &= (\mathtt{processK}|\mathtt{processT}|\mathtt{processT}|\mathtt{processT}|\\
&\quad \mathtt{processM}|\mathtt{processM}|\mathtt{processM})\\
V_A &= \mathtt{processV}\left\{{}^{skvaCh}/_{skvCh}, {}^{pkdaCh}/_{pkdCh}\right\}\\
V_B &= \mathtt{processV}\left\{{}^{skvbCh}/_{skvCh}, {}^{pkdbCh}/_{pkdCh}\right\}\\
V_C &= \mathtt{processV}\left\{{}^{skvcCh}/_{skvCh}, {}^{pkdcCh}/_{pkdCh}\right\}\\
D_A &= \mathtt{processD}\left\{{}^{skdaCh}/_{skdCh}, {}^{pkvaCh}/_{pkvCh}\right\}\\
D_B &= \mathtt{processD}\left\{{}^{skdbCh}/_{skdCh}, {}^{pkvbCh}/_{pkvCh}\right\}\\
D_C &= \mathtt{processD}\left\{{}^{skdcCh}/_{skdCh}, {}^{pkvcCh}/_{pkvCh}\right\}
\end{aligned}$$

As before, we will denote the left hand side P and the right hand side Q. For better readability concentrate on the important steps. After key distribution, the honest voters will execute the protocol with their trusted device and eventually output the following messages on `ch1`:

$$\begin{aligned}
P \quad &\to^* \ldots \to^* \xrightarrow{\mathtt{out}(ch1,x_1)} \xrightarrow{\mathtt{out}(ch1,x_2)}\\
&\nu r_1.\nu r_2.\nu r_3.\nu r_4.(P_1|\\
&\left\{{}^{(penc(a,pk(skt),f(r_1,r_2)),sign(sign(penc(a,pk(skt),f(r_1,r_2)),skda),skva))}/_{x_1}\right\}|\\
&\left\{{}^{(penc(b,pk(skt),f(r_3,r_4)),sign(sign(penc(b,pk(skt),f(r_3,r_4)),skdb),skvb))}/_{x_2}\right\})
\end{aligned}$$

$$\begin{aligned}
Q \quad &\to^* \ldots \to^* \xrightarrow{\mathtt{out}(ch1,x_1)} \xrightarrow{\mathtt{out}(ch1,x_2)}\\
&\nu r_1.\nu r_2.\nu r_3.\nu r_4.(Q_1|\\
&\left\{{}^{(penc(b,pk(skt),f(r_1,r_2)),sign(sign(penc(b,pk(skt),f(r_1,r_2)),skda),skva))}/_{x_1}\right\}|\\
&\left\{{}^{(penc(a,pk(skt),f(r_3,r_4)),sign(sign(penc(a,pk(skt),f(r_3,r_4)),skdb),skvb))}/_{x_2}\right\})
\end{aligned}$$

The attacker can now target e.g. voter V_A and copy his vote. Note that he can identify which voter cast which ballot as they are signed and the keys publicly available. Through $V_C^{c_1,c_2}$ he submits $penc(a, pk(skt), f(r_1, r_2))$ in the left hand case or $penc(b, pk(skt), f(r_1, r_2))$ in the right hand case to D_C and obtains $sign(penc(a, pk(skt), f(f(r_1, r_2), r_5)), skdc)$ and $penc(a, pk(skt), f(f(r_1, r_2), r_5)$ or $sign(penc(b, pk(skt), f(f(r_1, r_2), r_5)), skdc)$ and $penc(b, pk(skt), f(f(r_1, r_2), r_5)$ respectively, where r_5 is a fresh name (nonce). He can then sign the message and publish it on the bulletin board (i.e. send in on channel `ch1`):

$$(penc(a, pk(skt), f(f(r_1, r_2), r_5)),\\ sign(sign(penc(a, pk(skt), f(f(r_1, r_2), r_5)), skdc), skvc)) \quad (16)$$

or in the right hand case

$$(penc(b, pk(skt), f(f(r_1, r_2), r_5)),\\ sign(sign(penc(b, pk(skt), f(f(r_1, r_2), r_5)), skdc), skvc)) \quad (17)$$

The mixers will then check the signatures - which are apparently correct - and the talliers will publish the decrypted votes. On the left hand side, we will obtain two votes a and one vote b, on the right hand side one vote for a and two votes for b. Thus the frames are not statically equivalent, hence both sides are not bisimilar. □